O'REILLY®



Introducing iOS 8

SWIFT PROGRAMMING FROM IDEA TO APP STORE

Steve Derico

目錄

iOS8 开发入门

作者：Steve Derico

译者：sing_change

来源：iOS开发初学者入门

赞助译者：



写在前面——为什么要推荐这本书

首先，这本书写的特别棒，深入浅出，讲解到位，我理解了不少以前不懂的知识点。非常感谢作者！

我发现国内不少人因为这是英文书籍的缘故，看这本书的人比较少，反而错过这么好的入门书。

其次，以文会友，我在看这本书的时候还是遇到了一些不懂的地方，希望能通过翻译来理解，翻译之后还不懂，期望能够遇到大牛指教。

最后，也想练习一下自己的英文。

要说明的是，我不是要把整本书翻译完，我觉得这样可能会侵害版权吧，所以我以读书笔记的形式来介绍，里面会有我自己的想法，当然了，会标注出来哪些是原文，哪些是自己的想法。一般我的想法会放在最后。

那么，我们来看看作者是怎么介绍他这本书的吧！

本书适合没有任何编程经验的人，使用的编程语言是swift。每章节分为两个部分：首先是1.课程部分，解释定义概念，说明某个知识点；接着就是2.练习部分，通过构建实际的APP来理解前面介绍的知识点。

Learn how to think differently-and see the worlds from a whole new perspective.

这句话也是我为什么要学习编程的原因，怕翻译不好，所以贴了原文。不是说懂了编程就能怎么样，而是学习了之后，能从不同的角度思考问题，从一个全新的角度来看世界。或者说，可以更好的理解这个世界一部分运行机制是什么样子。

全书一共有8大部分：

01 学习基本的编程构造

02 学习Swift编程语言

03 开发iphone和ipad的APP

04 在APP中使用GPS来锁定用户位置

05 在APP中拍照或者从相册中选择照片

06 在APP中使用分享到Facebook 和 Twitter技术

07 提交你开发的APP到苹果的应用市场

08 在苹果应用市场中管理和运营你的APP

OK，现在来介绍一下作者。Steve Derico，是Bixby Apps公司的创始人和领导者，为世界五百强企业（例如宝马、联想、MGM Resorts等等）开发APP。同时，作者也是AppSchool.com的创始人，是面向初学者在线编程教学网站，为毫无编程经验的人提供学习开发APP的课程。学校已经教授了几百个无编程经验的人学习如何开发APP。作者的Twitter是 @stevederico

出版商是O'REILLY，不少著名的编程书籍都是这个出版商出版的。

好啦，今天就先写到这里吧~

我尽量每天都写一点~如果对此感兴趣，欢迎订阅~

如果你觉得我的文章不错有些帮助，不妨打赏点上下班的公交车费吧~



本书在豆瓣上的介绍：http://book.douban.com/subject/26166179/

亚马逊的购买链接：http://www.amazon.cn/Introducing-iOS-8-Derico-Steve/dp/1491908610/

由于Swift语言本身还在变化和发展中，本书的勘误和最新更新内容可以在AppSchool.com/book找到

作者的联系方式：

Twitter:http://www.twitter.com/stevederico

Email:steve@appschool.com

Phone number: (415) 779-2771

作者的公司网址：http://www.bixbyapps.com

作者创建的初学者网站：https://www.appschool.com

前言

> 这是前言的一部分，我分成了2部分翻译，今天完成了前言的翻译。
>
> 老外写文章还没蛮啰嗦的，正是因为这么啰嗦，才适合初学者，有耐心，讲解详细到边边角角。
>
> 好啦，下面就是正文啦~

我为什么要写这本书

我是那些毫无编程经验的人写的这本书。目前市面上大部分的编程书籍都是建立在读者已经知道了一种编程语言，或者已经获取了计算机学位的基础假设之上，这样的假设给那些初学者增加了学习的障碍和困惑。

而这本书是为纯粹的初学者设计的，为进入IOS编程世界提供个人指导。本书去掉了复杂的术语，用生活中相关的实际例子取而代之，本书用一些常见的场景，例如去杂货店，开车，在餐厅里吃饭，来教授编程知识。每章节都会用清晰简明地讲解概念。

这本书的目标是讲解的内容从最初的概念一直到如何发布应用到ihpone的应用市场（APP store），你将会学到基本的编程知识概念，开发APP的步骤和过程，以及如何将APP发布到APP市场，无需编程经验即可阅读此书。You will learn to think differently and see the world from a new perspective.（你将学会如何从不同的角度思考，从一个新的角度来看世界。）这本书将给你一个工具来改变你和其他人的生活。别担心别害怕，这本书已经去掉了可怕的复杂编程术语了。

> 这后面原书中还有一大部分描述这书适合什么人阅读之类的，总结起来就是一句话，哪怕您从事的工作和互联网计算机没有关系，例如银行柜台、会计、销售等，只要您想学编程，这本书就是您最佳选择~
>
> 因此就略了，都肯来看我的译文了，肯定是想学编程了，就不需要看作者的这番鼓励话语了。

浏览这本书

本书使用的编程语言是苹果公司2014年发布的Swift，截屏是使用的OS X Yosemite（10.10）。在学习此书时，注意英文的大写和拼写，这些非常重要，不能写错了。

Chapter 1, Getting Started

第一章，准备开始

包括：应用市场基本介绍，设置你的成功之路，把Xcode安装到你的电脑。

Chapter 2, Introduction to Programming

第二章，编程介绍

包括：编程的基本介绍，MVC模式（*Model-View-Controller*），构建你的第一个APP。

Chapter 3, Diving into Swift

第三章，了解**Swift**

学习Swift的基础知识，变量类型，集合，循环。建立一个小费计算机应用。

Chapter 4, Diving deeper

第四章，更深入了解**Swift**

学习方法、对象、类。建立一个赛车应用。

Chapter 5, Building Multiscreen Apps

第五章，建立多个页面的**APP**

学习视图控制器（view controller）、表视图（table views）和导航控制器（navigation controller）。建立一个你自己的护照APP。

Chapter 6, Next Steps : Debugging, Documentation, and App Icons

第六章，下一步：调试，文档和应用图标

学习如何解决bug，学习如何使用官方文档，改进护照APP。

Chapter 7, Devices and Auto Layout

第七章，设备和自动布局

学习如何使用自动布局（Auto Layout）功能来为APP设置不同尺寸的布局设计。改进护照APP。

Chapter 8, Maps and Location

第八章，地图和定位

学习使用使用GPS功能来定位用户的地点，生成地图和地图上的红点（plot points）。将地图功能加入到护照APP里。

Chapter 9, Camera, Photos, and Social Networks

第九章，相机，相册，社交网络

获取使用摄像机、浏览手机中图片的权限，增加分享到Facebook和Twitter的功能。创建一个名为“自拍”的APP，使用前置摄像头拍摄照片。

Chapter 10, Running on a Device

第十章，在设备上运行**APP**

学习如何在IOS设备（真机）中运行开发的APP，开发自拍应用

Chapter 11, Submitting to the App Store

第十一章，提交到**APP**市场

如何设置必须的合同，生成APPsotre清单，提交你的APP

Chapter 12, Managing and Marketing Your App

第十二章，管理和运营你的**APP**

APP上线后如何管理APP，如何更新和维护你的APP

Appendix A

附录**A**

什么是Objectvie-C，为什么你需要知道Objectvie-C，以及如何阅读Objectvie-C代码。

阅读之前需要准备好：

一台苹果电脑

运行OS X Mavericks（10.9）以上系统。

一台**IOS**设备

可以是iPhone，iPod Touch或iPad。IOS设备需要运行IOS7及更新的系统。

专注的工作空间

学习开发APP与学习历史相比是截然不同的学习模式，编程需要你的大脑换一个思考方式和思维方式，需要绝对的专注，不能分心。当你开始阅读这本书的时候，请关闭的你浏览器，Facebook，Twitter（微信朋友圈、新浪微博）和Instagram。每天能拿出特定的时间来写代

码，只开着Xcode，专心学习。

积极的态度

一开始学习开发可能会很难，但是当你开发出人们喜欢的APP，为人们的生活创造了价值后，你就会觉得之前的付出是多么值得的。当学习开发遇到困难时，请记住：你可以做到，没有什么不可以。坚信你能行，更加勤奋。

书中的一些固定用法

本书中的字体有自己的含义：

斜体

表示新的术语定义，URL，邮件地址，文件名，文件扩展名

等宽字体

用于表达程序代码，或者表明程序元素，例如变量名、函数名、数据库、数据类型、环境变量、声明以及关键词。

由于简书中只能显示一种字体，所以我用引用的格式代替了。

等宽粗体

表示命令，或者应该由用户输入的文本

由于简书中只能显示一种字体，所以我用引用的格式+加粗代替了。

等宽字体斜体

表示此文本应该由用户提供的数值代替，或者根据内容决定的数值。

由于简书中只能显示一种字体，所以我用引用的格式+斜体代替了。

>  这个图标表示小贴士、建议或普通的笔记
>
>  这个图标表示警告或者注意事项

本书的代码都可以从这个网站中下载到：http://appschool.com/book/

前言部分结束。

> 一开始没有打算把附录拆分翻译的，但是没有想到翻译起来速度太慢了，所以还是分开吧，几乎晚上没有做什么其他的事情，看来还是刚刚开始，速度跟不上，以后翻译多了，熟能生巧，就会快一些了，尽量不拆开翻译。翻译、校对，然后发布到简书调整格式，用了不少时间。
>
> 如果你发现我文章中有错别字，请留言指出，或者邮件我：sing8796185@163.com。我在看到您的指正后会第一时间改正错别字。输入法打字，如果出现了错别字，还请谅解。文章中如果有翻译不对的地方，请尽量指出，希望能借此结识更多编程高手~

第一章：准备开始

在这一章里，为你的学习做好计划和准备。先了解APP Store，设置你的Mac，为开发IOS程序做好准备。

做好规划

在启程前先花一点时间为旅程做好规划，确保完成下面的事项：

了解***App Store***

当你学习开发APP时，确保利用好手头已有的资源：App Store。你能从App Store里看到目前的趋势，看到最受欢迎的APP的特点。每天都去看一下 排行榜，然后下载新的APP，看一下目前人们喜欢什么样的APP，不喜欢什么样的APP，了解这些能给你带来更多的竞争优势。

解决你自己的需要或者遇到的问题

很多伟大的企业家一开始构建某个产品都是为了解决自己的需求，开始构思APP之前，先考虑一些你希望出现的APP，然后去做。为解决个人需要而产品的激情和思考，将会使你开发出杰出的APP，并能解决你遇到的问题。

观察人们的习惯

下次你在坐公交车或者在咖啡厅喝咖啡的时候，看一下周边的人在用什么APP，从中你可以知道目前的趋势。

熟能生巧，多多练习

你开发的第一个APP不会是你最好的APP，不要被这个结论吓到，也不要被其他有些的APP吓到，。Facebook的办公室里有一张巨大的海报，上面写着：“Move Fast and Break Things”（快速前进，打破常规）。正是这张海边激励着Facebook的员工不去惧怕失败。软件有个优势，就是你总可以推送一个新版本。

> Page 1 | Chapter 1 : Getting Started
>
> 看来一天一页的进度才是刚刚好的进度，一周一章也太夸张不好控制，一天翻译一页刚刚好~
>
> 明天继续。加油！

编程之道，多看多敲（***Ship early, ship often***）

多看多敲是学好程序的关键。开发者在App store里看到了好的东西后，能够融会贯通。什么是你APP的核心呢？问一下自己下面几个问题，“这个应用提供了什么价值？这个新特性能够让更好的展现APP的价值吗？我们是不是必须要增加这个新特性？”这些问题能够帮助你确定某个功能在第一版出现还是在以后的某个版本迭代，为自己的APP功能开发列出优先级顺序。

积极的态度

一开始学习开发可能会很难，但是当你开发出人们喜欢的APP，为人们的生活创造了价值后，你就会觉得之前的付出是多么值得的。当学习开发遇到困难时，请记住：你可以做到，没有什么不可以。坚信你能行，更加勤奋。

> 这段话在之前出现过了，看来这个作者喜欢一个观点来回描述好多次啊！直接复制过来的节奏啊。

App Store

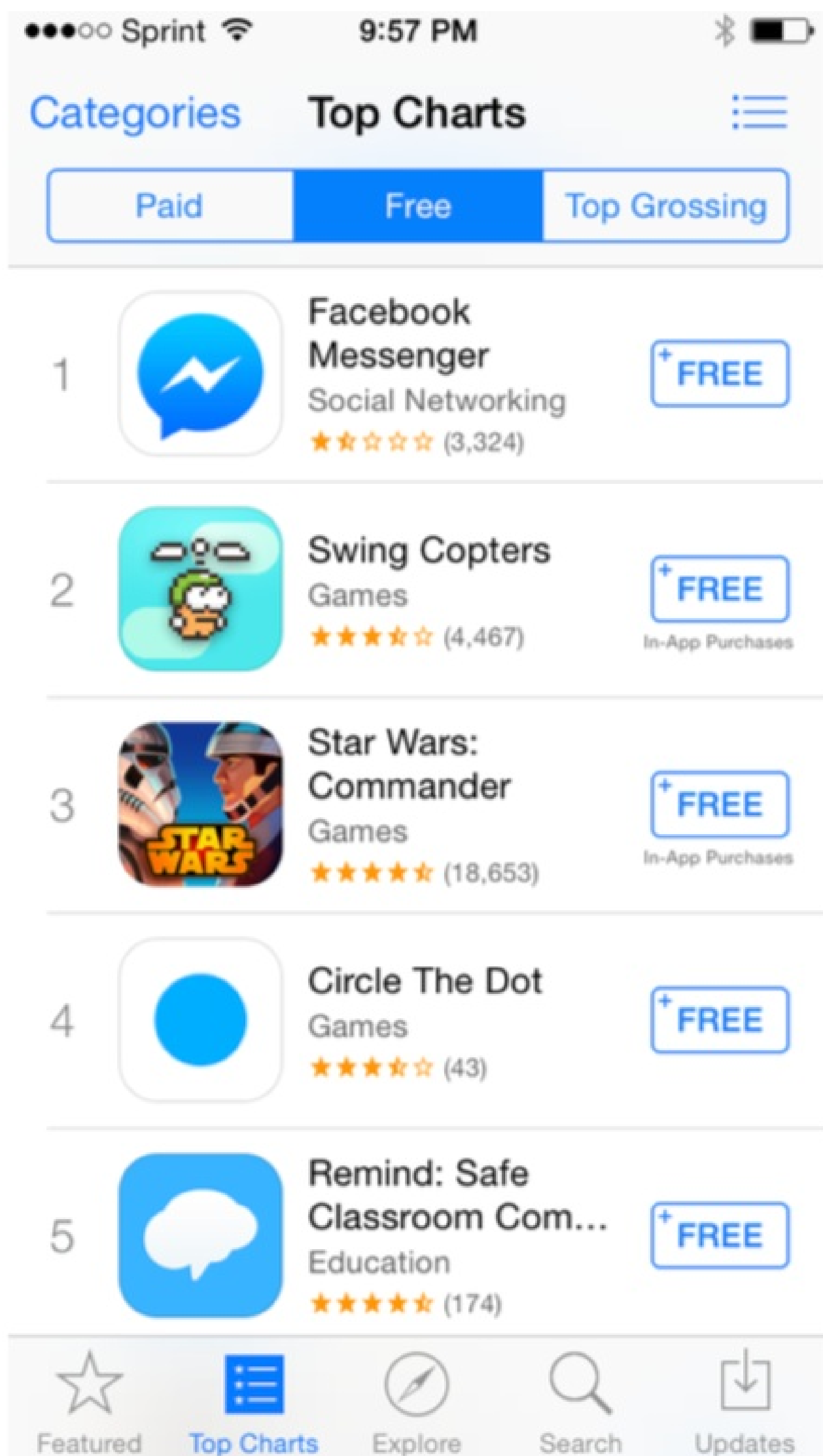
IOS是苹果手机运行的系统，支持的设备有：iPhone，iPad，iPod Touch。iOS应用市场在2008刚刚年刚刚出现的时候，只有500个应用。经过六年的发展，现在已经有了超过120万个应用。这些应用的下载量超过了750亿次。在iOS设备上，我们可以通过App Store应用进入应用市场，在Mac电脑上，可以通过iTunes进入应用市场。

> 原文的布局是这样的：上面先出现一段文字解释一张图片，这张图片会出现在下一页。这样看起来好乱，所以我在这里还是按照中国人的习惯改一下。
>
> 现在的布局：先上图，再解释图的意思。图下方的文字就是解释上方的图。



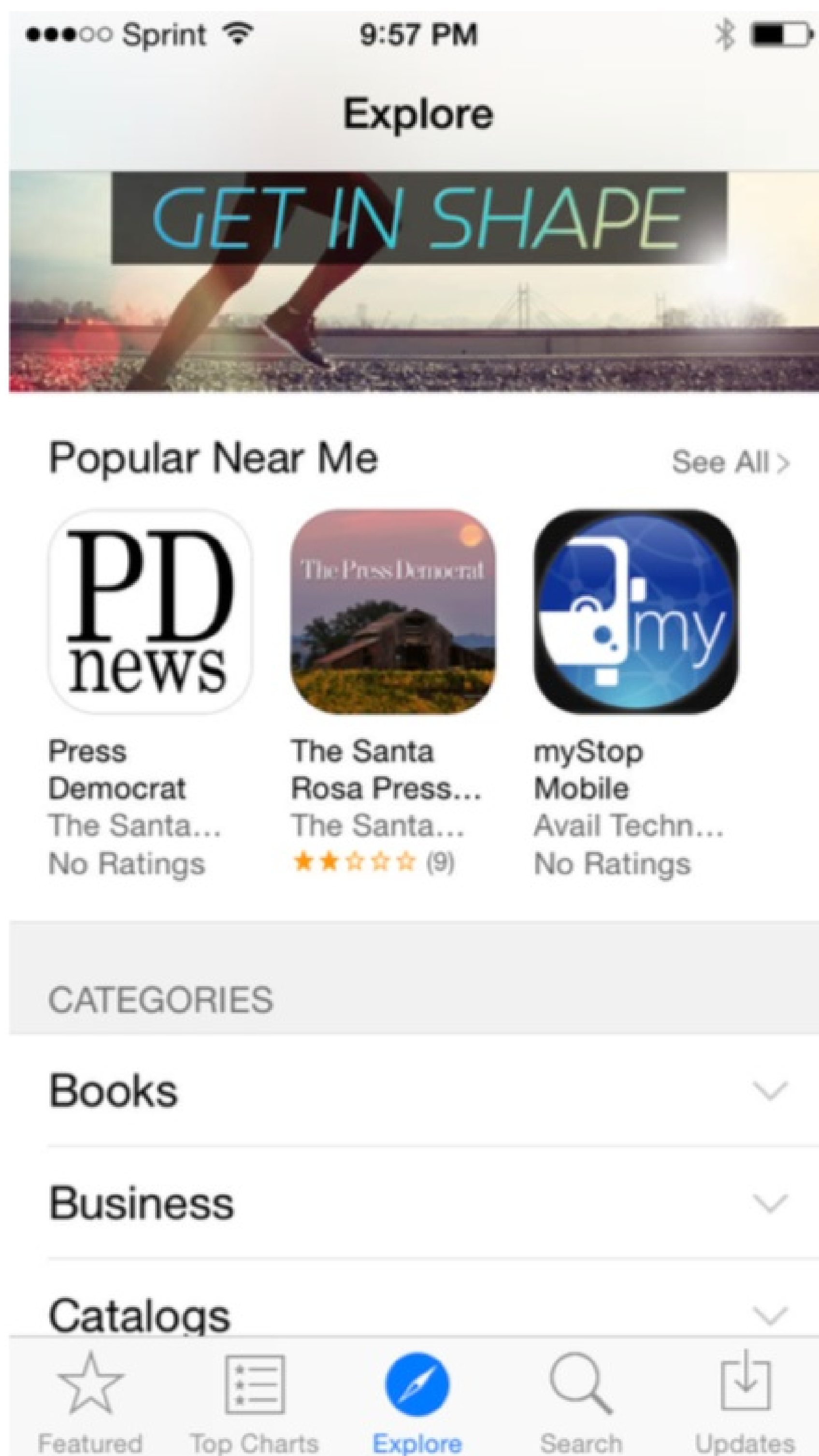
在图*1-1*中，底部一栏中的Featured（精品推荐）是苹果官方筛选出的精品应用，这些应用荣登榜单的原因多种多样，比如杰出的设计，新奇的想法，使用了新的技术等等。默认显示全部类别的精品应用，你也可以通过点击左上角的“Categories（类别）”来查看具体某个类别的精品应用。

> Page 2 | Chapter 1 : Getting Started



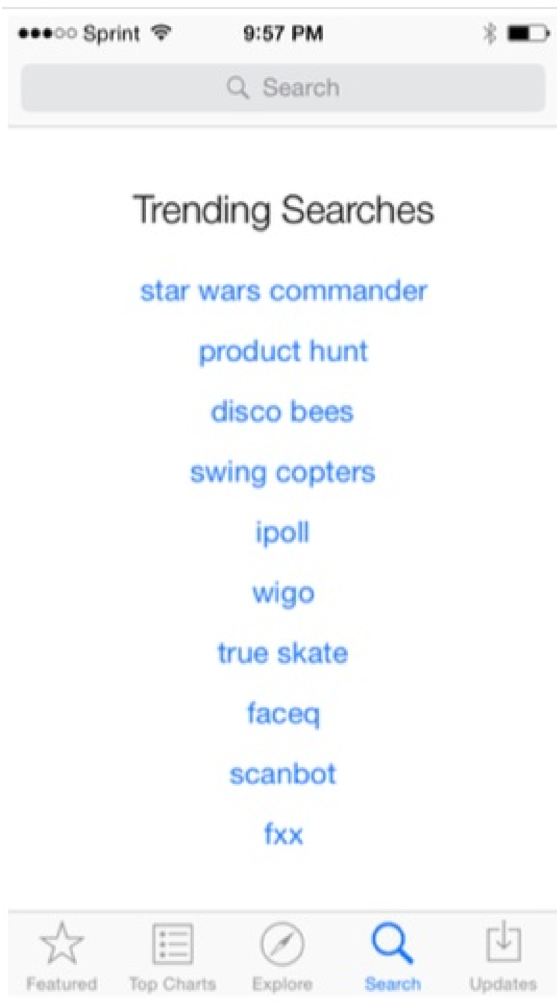
在图*1-2*中，底部一栏上的Top Charts（排行榜）显示的是App Store上最受欢迎的APP，包含Paid（付费应用）、Free（免费应用）和Top Grossing（热门畅销应用）的下载量各前150名。Top Grossing（热门畅销应用）指的是获得收入最多的APP。Top Grossing（热门畅销应用）大部分的应用都是免费的，通过应用内付费的形式来获得收入，应用内付费是指，用户可以在某个APP内购买需要虚拟物品。

The App Store | Page 3



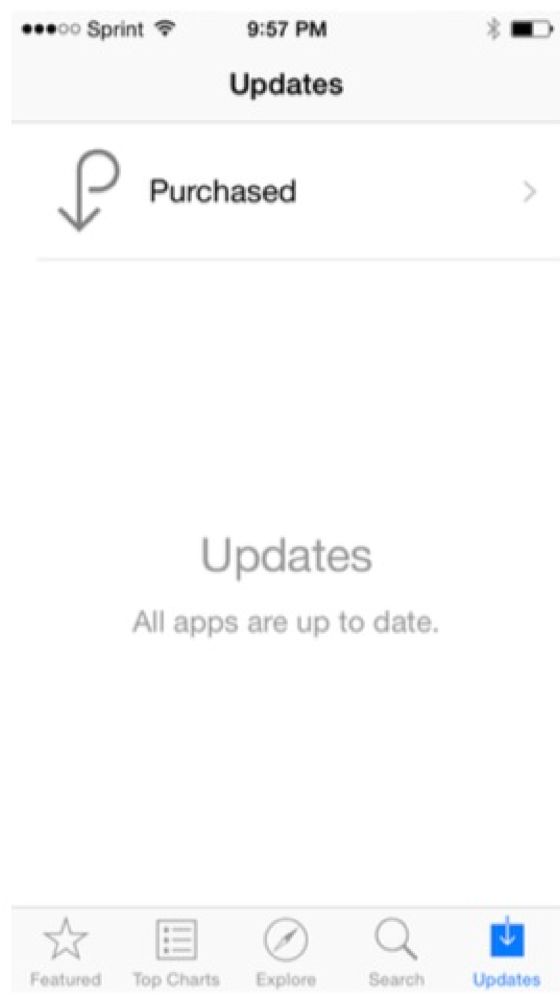
在图*1-3*中，底部一栏上的Explore（探索）提供了另外一种方式来浏览App Store，Explore（探索）允许你浏览一个特定的类别，找到相关的应用程序。例如：在财经类别下有很多的子品类，像是精选财务，资金管理，银行等。Explore也会根据你的地点来推荐APP，例如纽约的地铁APP，旧金山的市政公债APP。

> Page 4 | Chapter 1 : Getting Started
>
> 这本书英文版本有298页，如果一天翻译一页的话，需要10个月的时间，那进度也太慢了吧，看看有什么办法能加快进度否。
>
> OK，今天到了第四页了，明天继续，加油加油！



在图*1-4*中，底部一栏上的Search（搜索）是大部分用户查找APP的地方，点击Search（搜索选项卡）后，会显示一排经常搜索的清单，清单里包括最常用的搜索，顶部的搜索框与我们常见的搜索框是一样的，搜索完成后显示搜索结果。在iOS8里，同时还会显示每个APP的两张截图。

> The App Store | Page 5



在图*1-5*中，底部一栏上的Updates（更新）显示的你已经拥有的APP的更新情况。在iOS8里，App Store会自动更新你的应用。点击顶部的Purchased按钮，可以重新下载你在其他设备上或者删除的应用。

> Page 6 | Chapter 1 : Getting Started

如何安装**Xcode 6**

开发iOS应用必须安装Xcode。Xcode是苹果官方的开发软件，可以用来开发iOS系统和OSX系统的应用，可以免费下载安装到你的Mac上，只能在Mac电脑上安装运行，无法在Windows、Linux系统的电脑或者iPad上使用Xcode。Xcode和Word软件相似，只不过Xcode是用来编程的，而Word是用来写东西的。Xcode提供了非常多有帮助的功能来确保你的代码能够正常运行。在进行开发之前，确保完成下方的事情。

> How to Install Xcode6 | Page 7

你需要做的事情：

一台运行***OS X Mavericks***（***10.9***）或者更高系统的***Mac***电脑

OS X Mavericks（10.9）可以免费更新到你的Mac上，如果你的Mac系统还没有更新，请到这里完成更新：https://www.apple.com/osx/。

一个***App Store***帐号

App Store帐号也是你的Apple ID，就是用来购买应用时需要输入的邮箱和密码，这个Apple ID也可以在iTunes里购买歌曲，登录iCloud。帐号可以在这里免费获得：https://appleid.apple.com

你的***Mac***电脑的管理者密码

最后，安装Axcode，你需要输入Mac的管理者密码，这个密码和Apple ID不一样，是用来登录Mac电脑的。如果你无需登录即可进入Mac电脑，那么你已经是管理员了。如果你的Mac是公司的电脑，你也需需要格外的权限。

> *Tip* 小贴士：__查看是否是管理员的方法：点击屏幕左上角的苹果图标，点击"系统偏好设置—>用户与群组"，然后你就可以看到自己目前是以什么身份登录的系统，你目前是否拥有管理员权限。

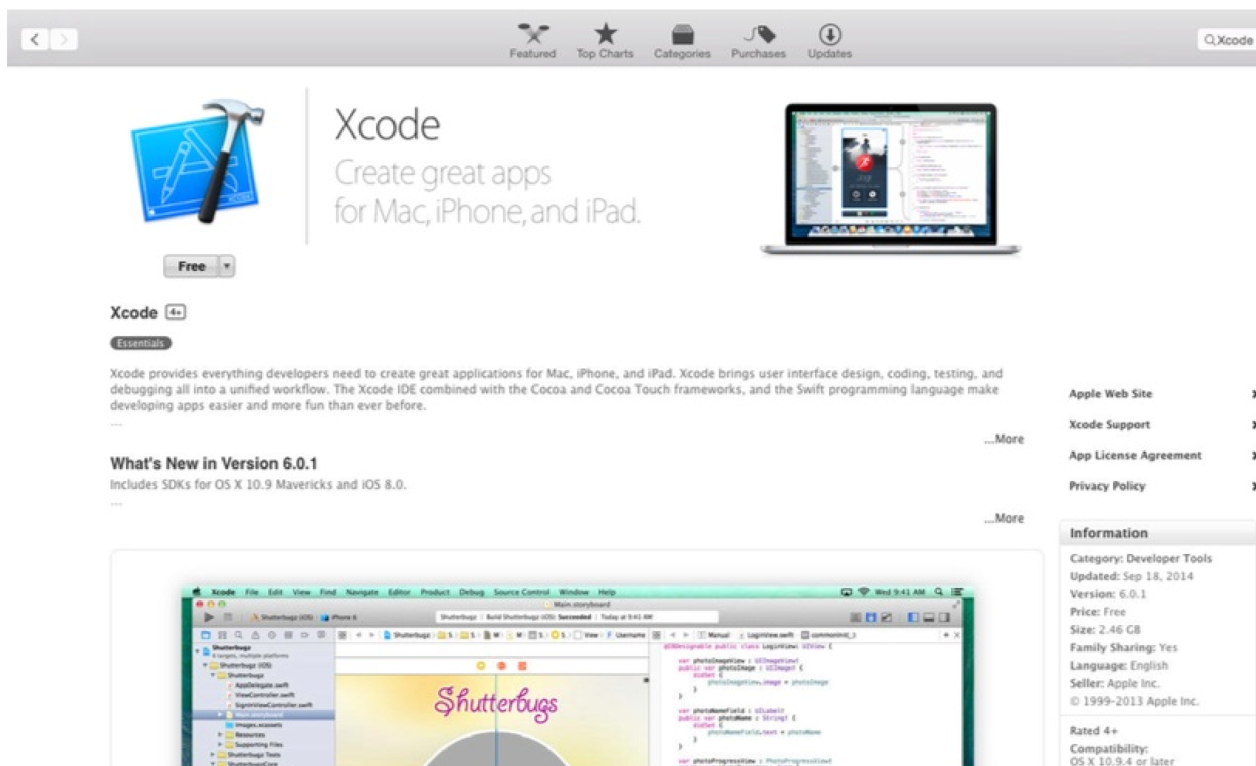
Xcode可以在App Store里下载，大约超过4GB，在下载Xcode之前，确保你的Mac至少拥有5G的空间，下载过程大约持续一个多小时（由网络情况而定）。

如果你还没有开始下载，请点击这里：Xcode页面，或者在浏览器中输入下面的网址：

https://itunes.apple.com/us/app/xcode/id497799835?mt=12

Mac上的App Store应该会自动打开，如果没有，请参照图1-6，点击网页中的"View in Mac App Store"按钮，即可转向Mac的App Store。

> Page 8 | Chapter 1 : Getting Started



参照图*1-6*，接着点击银色的“Free”按钮，接着点击绿色的“Install（安装）”按钮，输入你的App Store帐号。

这样，下载就开始了。点击顶部的“Purchases（已购项目）”你可以看到下载进度。

点击“Purchases（已购项目）”后，Xcode软件旁边的按钮变成“Open（打开）”时，说明下载已经完成了。接着点击“Open（打开）”按钮，Xcode开始安装。

在安装过程中，你可能会看到一个警告，显示Xcode的用户协议，阅读这个协议，然后做出你的选择。

下一步就是马上要安装到系统上了，点击“Install（安装）”。

接着会提示你登录输入密码，这个帐号密码是登录Mac电脑的密码，不是App Store帐号密码。

输入完帐号密码后，安装进程开始，进度条满格后，Xcode就安装好了。

如果Xcode没有像你想象中那样安装成功，不比担心。学习的最好方法就是发现解决错误。熟能生巧嘛~

> How to Install Xcode6 | Page 9
>
> 太好啦，第一章终于翻译完啦，好高兴啊~
>
> 第二章就真正开始了解程序了，这一章还是介绍工具。工欲善其事，必先利其器。
>
> 如果你发现我文章中有错别字，请留言指出，或者邮件我：sing8796185@163.com。我在看到您的指正后会第一时间改正错别字。输入法打字，如果出现了错别字，还请谅解。文章中如果有翻译不对的地方，请尽量指出，希望能借此结识更多编程高手~

第二章：编程介绍

编程看起来像是一件令人望而生畏的事情，但是不一定这样。大部分的编程都可以归结为一些基本的数学技巧。不管过去你的数学水平如何，现在，你只要学习一些基本技巧就可以开始了。在你开发APP之前，需要做的很重要的一件事情就是，理解神奇的表象背后的原理。在这一章节中，我们将学习编程的基础知识，了解对象是如何建立，以及一些保持代码干净整洁的最佳实践。在学习完这些之后，就可以将所学知识付诸实践，构建第一个APP了。

学习编程就像是学习骑自行车，起初，看起来好像永远做不到，而且完全不知道如何去做。这不说明你不具备基本的骑自行车的技术，而是到目前为止，你还不知道如何使用这些技术。编程就是教你的大脑用一种不同的方式来思考事情，而基本的技巧没有什么不同，只是概念是新的。

基础构件（Building Blocks）

我们将要明确定义一种编程语言的基本要素，都是一些新的概念定义，在学习过程中，会经常看了再看学了再学，初看起来像是在学习一种外语，但是学习骑自行车也是这样。

变量（ Variables）

当你检查银行账户余额时，上面显示你现在可用的金额。你的银行账户余额在月初可能是100美元，在你发工资的那天可能是350美元。鉴于账户上的金额是在变的，“账户余额”一词通常表示现在的数额。这是一个典型的变量，变量代表一个值。变量有很多种形状和大小。不同类型的变量代表不同的值。标量可以是数字、字母、单词、真、假，甚至是一个定制轿车。

> Variables | Page 11

整型（Integer）

一个整型类型是一个整数，没有小数，正数和复数均没有小数：

```
-10, 0, 100, 21031
```

整型类型使用场景有，电影评分的几颗星，住所所在的门牌号，或者球队的得分。

浮点型（Float）

有时候我们需要对数值有更精确的描述。举个货币的例子，小数常常用在表达不足一美元时美元后面的美分：

```
$10.51
```

有小数的变量我们成为浮点型。浮点型也是小数的另一种叫法。

布尔类型（Boolean）

如果有人问你天空是否是蓝色的，你会用“是”和“否”来回答，不能回答成$103.45或者“香蕉”。“是”和“否”就是布尔类型。布尔类型就像是电灯的开关，要么是“开”，要么是“关”，“真”或“假”，没有中间状态。

字符串类型（ String）

有人问你的名字，你会用一串字母构成的单词回答（这里说的是英语名字，译者注）：

```
"Steve Derico"
```

字符串用来表示一串字母构成的词或者句子。一个字符串里可以包含字母，数字和符号，用一堆引号包括起来。例如：

```
"Steve is cool."
     "Where is the ball?"
     "Go Giants"
```

类 （Classes）

看一下拥挤的道路，你会看到各种各样的机动车，有SUV，跑车，卡车，厢式车。每种车看起来都不同，但每一辆车都具有一些核心特性。每一辆车都有轮子、发动机、和刹车。每种类型的车，不管是什么牌子，不管是什么类型什么风格的车，都一定会有这些核心特性。没有这些核心特性，就无法成为一辆车。这些核心特性叫做“属性（attribute）”

> Page 12 | Chapter 2 : Introduction to Programming

一辆车不仅仅是一堆金属，它存在是有目的的，它为消费者提供了价值。一辆车可以驾驶，可以发出汽车喇叭声，可以刹车，可以转向，这些基本的使用方法在每一辆车上都存在。没有这些使用方法，就不能成为一辆车。这些核心的使用方法被称为“行为（behavior）”。

如果你打算设计一辆新车，一张设计图会是一个不错的开始。设计图是一份文档，作为一个模板来构建一些东西。设计图定义了这辆车的属性和行为，一个基本的车辆设计图可能类似于表2-1。

```
表2-1 车辆设计图
 - 有轮子
 - 有发动机
 - 有刹车
 - 可以前进
 - 可以停车
 - 可以左转右转
```

带“有”字的是属性，带“可以”的是行为。现在你可以在工厂里使用这份设计图生产汽车了，每一辆车都会具备设计图中所列出的属性和行为。

而一个类，是虚拟物品的设计图。类定义了所需的属性和行为，就像是曲奇成型器，一个类（曲奇成型器）可以从一个设计图中制作出无数的物品（曲奇），一个星星形状的曲奇成型器，可以制作出无限的星星形状的曲奇，所生产出的每一个曲奇，都具有和曲奇成型器相同的形状。

对象（Object）

如果所有的汽车都有相同的属性，那是什么让一辆车与众不同？是属性的值让每一辆车与众不同。你可能见过一辆绿色的旅行车在路上行使，这辆车具有标准的轮子，标准的柴油发动机，标准的刹车系统（见表2-2）。接着你可能看见一辆红色的跑车，有大号轮子，功能强大的发动机，性能出色的刹车系统（见表2-3）。这两辆车具有相同的属性却有不同的值。

> Objects | Page 13

表**2-2**	绿色旅行车
轮子	标准
发动机	柴油机
刹车系统	标准
加速系统	差
停止	强
转向	一般

表2-3	红色跑车
轮子	大
发动机	大
刹车系统	性能良好
加速系统	极好
停止	极好
转向	极好

每一辆车都是车这个类下的对象，类制作生产出对象。对象具有这个类的属性和行为，在这个例子中，对象具有车这个类的属性和行为。单词实例（instance）和对象（object）是同义词，先不要关注这点，因为人们倾向交替使用这两个词。

> 更多信息可以参考附录B中的同义词列表。

你是人类的一个实例，世界上再也没有第二个你，即便是你有一个双胞胎兄弟（姐妹），每个人都是独一无二的。眼睛的颜色、头发的颜色、姓名，所有的人都有这些属性。例如，参见表2-4和表2-5，属性列在左边，值列在了右边。

表2-4	路人甲
眼睛的颜色	蓝色
头发的颜色	棕色
姓名	Larry

表2-5	路人乙
眼睛的颜色	棕色
头发的颜色	棕色
姓名	Magic

正是属性后所跟的值早就了独一无二的对象（Object）。人类拥有几百个属性，每个属性对应几千个值。属性和值的排序是无尽的，所以，每个人和另外的其他人们相比，都是独一无二的。

> 个人挑战：对人类来说，你还能想到哪些属性呢？

方法（Method）

每天早晨起床后，你都会做一些惯例的事情：离开床，洗个澡，刷牙，穿衣，吃早饭，出门。这些例行步骤特别像一台计算机的运作机制，计算机会执行例行步骤清单来完成任务，这个例行步骤清单叫做方法（Method）。一堆能完成一个特定的任务的代码集合在一起，便是一个方法。如果你要写一个早晨例行清单，可能看起来是这样的：

```
1\. 醒过来
2\. 离开床
3\. 冲个澡
4\. 刷牙
5\. 穿衣
6\. 吃早饭
7\. 出门
```

这个方法将会在每天早上你起床后执行，当这个方法执行完毕后，你已经准备好开始新的一天了。方法执行完毕后产生的结果叫做输出（Output），而输入（input）是你放入方法中的东西，例如，对菜谱来说，输入是配料；在制作成纸张之前，输入是树木；而在这个案例中，你还未睡醒的身体是输入。

> Methods | Page 15

```
个人挑战：写下制作花生酱、果冻、三明治的步骤，看看你能写到多么详细。
```

继承（Inheritance）

孩子会具有和父母相似的面部特征，“她有她爸爸的眼睛”，“她的嘴巴长得像妈妈”，孩子出生后，具有父母双方基因和外部特征，父母的属性和行为传递到了孩子身上。父母和孩子可能具有相同的属性，例如头发的颜色，眼睛的颜色，或者肤色。父母和孩子也可能具有相同的行为，例如都擅长某项体育运动。Ken Griffey, Jr 和 Ken Griffey, Sr 父子二人，都具备职业棒球手的职业道德和身体特征。 父类传递给子类属性和行为被叫做继承（Inheritance）。对一个类来说，继承能够让类可以扩展或者重构父类的功能。假设你创建的“车”类（下方表2-6）是“SUV”类（下方表2-7）的父类，“SUV”类将会继承“车”类的所有属性和行为，同时，“SUV”类还可以拥有只属于自己的属性和行为。

表2-6	车
轮子	标准
发动机	标准
刹车系统	标准
加速系统	
停止	
转向	

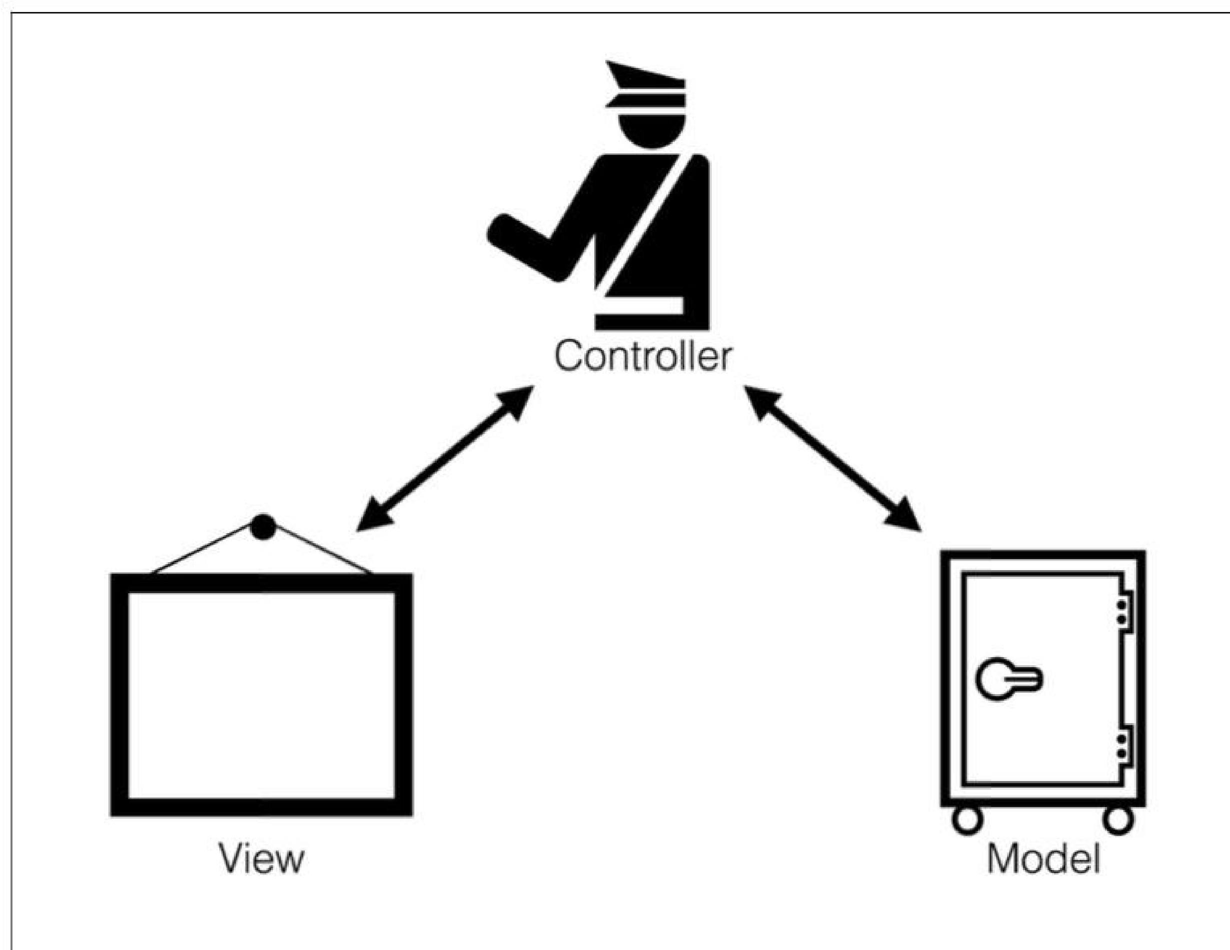
> Page 16 | Chapter2: Introduction to Programming

表2-7	SUV
轮子	泥浆轮
发动机	V8
刹车系统	标准
驱动系统	四驱驱动
加速系统	
停止	
转向	
能够山路坡道行驶	
能够牵引船只	

一辆SUV有轮子、刹车和发动机，但是它也可以重写自己继承来的属性。（override的意思是覆盖、重写、重构）重构（Overriding）这种能力可以改变子类的属性和行为。这就允许SUV类可以定制和控制所继承的属性和行为，我们注意到了，SUV类有发动机，但是是V8发动机，SUV类也有轮子，但是用泥浆轮代替了父类（车类）中的标准轮子，同时SUV类也有自己的独有的属性和行为，例如：四驱驱动系统，能够山路坡道形式的行为，能够牵引船只的行为。在计算机专业英语中，子类的英文是subclass。

模式，视图，控制器（Model, View, Controller）

壁橱（衣橱）是你存放衬衣、裤子、鞋子的地方。如何你能把所有的物品分类放在不同的区域，这样壁橱的使用效果最好，当你急需一件衬衣的时候，你不需要翻箱倒柜的找整个壁橱，无需在鞋子和裤子区域来找衬衣。保持壁橱分类整齐才能更好更方便的使用壁橱。同样，如果你想更换你的衬衣，你可以迅速更换无需碰触裤子和鞋子。 这样的道理同样适合编写代码。比起一堆乱似老鼠窝的代码，整齐有序的代码会指数倍的节省你的时间，当你必须回到以前的代码中查找修改一段代码时，整齐的代码能够更容易找到，改变坏习惯难于增加新习惯。模式-视图-控制器（Model, View, Controller）这样的组织结构，将你的代码分为3个不同的部分（见图2-1）。



Model, View, Controller | Page 17

模型（Model）

模型部分的代码和数据库有关。例如，你打算开发一个能够存储朋友联系方式的APP，那么，模型部分将会存储电话号码和地址等信息。记住，模型（Model）就是数据，把模型想象成装满0和1的保险箱（图2-2）吧。



Page 18 | Chapter2: Introduction to Programming

视图（View）

视图部分的代码和用户界面有关。视图部分就像是一个相框，能够展现一张图片，但是不知道展现的是什么图片（图2-3）。你的视图代码不应与正在展现的内容有联系，这种情况下，即使界面改变了，也不会影响内容。



Model, View, Controller | Page 19

控制器（Controller）

控制器部分的代码是和逻辑做决定有关。控制器就像是交通警察，他引导人们往哪里走（图2-4）。控制器直接与视图和模型沟通。控制器负责回应点击屏幕，在接收到点击屏幕这个指令后，控制器从模型中读取数据，然后告诉视图展现什么内容。而模型和视图永远都不会直接交流，所有的沟通都要通过控制器。



Page 20 | Chapter2: Introduction to Programming

练习：写一个Hello World程序

现在是时候来写你的第一个特别简单应用程序了，在这个练习中，你将会编写一个Hello World应用程序（图2-5）。这个应用程序有一个按钮（Button），当按钮被点击后，会在屏幕上出现Hello World这两个单词（图2-6）。





Exercise: Hello World | Page 21-- Page 22

第一步是打开Xcode。如果你在Dock中没有看到Xcode，点击右上角的搜索（Spotlight），输入Xcode然后点击第一个选项（图2-7）。



Exercise: Hello World | Page 23

现在你能看到Xcode出现在了Dock中。右键点击Xcode，在选项（Option）中点击“在Dock中保留”。 Xcode启动完成后，会出现一个欢迎界面。 点击“Create a New Xcode Project”，接下来，你会看见项目模板提示框（图2-8）。选择Single View Application，点击Next。接下来，你需要给你的项目填写一些信息（图2-9）。

Page 24 | Chapter2: Introduction to Programming





第一行是Product Name，也就是你项目的名称，这个名称将会成为你项目的名称以及存储这个项目的文件夹的名称。输入 HelloWorld作为项目名称。第二行是Organization Name，也就是开发这个APP的公司或者个人的名称。目前，你可以输入你的姓名，中间不要有空格。第三行Organization Identifier是用来生成bundle identifier的。bundle identifier就像是你APP的社会保险号，这是独一无二的标识，将你的APP与其他的APP区别开来。目前，你可以输入你的姓名，中间不要有空格。 最后两项非常重要，在这本书中，你将会使用苹果公司新的编程

语言：Swift。Swift发布于2014年六月，作为iOS和OSX编程语音。Swift拥有很多现代语言的特性，将会给程序员带来很多便利。将第五行的Language选择为Swift，这样就能确保你会一直使用Swift来编写这个项目。最后，Devices选择iPhone，不勾选Use Core Data。我们会在第七章的时候，重选设备（Devices）选项。 填写完这些信息后点击Next。

接下来出现一个对话框让你选择将代码存放在哪里（图2-10）。为你的所有项目专门新建一个文件夹，点击左侧下方的New folder按钮，给新文件夹命名为“Programming”，然后点击创建（Create）（图2-11）。最后，不勾选“Create Git Repository on My Mac”，然后点击创建（Create）（图2-12）。







> Exercise: Hello World | Page 25 - Page 27

Xcode项目打开了。 左侧的侧边栏叫做Project Navigator（图2-13）。Project Navigator有些像电脑上的文件夹，能让你方便简单的查找打开你项目中的文件，Project Navigator的文件夹叫做groups，它们并不是真的在你Mac文件夹系统里，只是打了个比方。



Xcode中间的这部分叫做Editor。你在左侧的Project Navigator选择了哪个文件，Editor就会呈现相应的内容。图2-14中的Editor展示的是项目细节，因为左侧的Project Navigator一栏里，这个项目文件被选中了。



> Page 28 | Chapter2: Introduction to Programming

右侧的侧边栏叫做Inspector（图2-15）。Inspector显示的是Editor中展现内容的可改变属性。所以Inspector中的属性是随着Editor内容的变化而变化的。



> Exercise: Hello World | Page 29

Xcode顶部的这一部分叫做Toolbar（图2-16）。看起来有些像iTunes的Toolbar。在Toolbar的左上角，有Play按钮和Stop按钮。这两个按钮是用来运行测试你的APP的。Toolbar中间的部分，你会看到Activity Viewer，当你的代码在编译的时候，你可以在Activity Viewer中看到进度。在Toolbar最右侧有两组按钮。



> Page 30 | Chapter2: Introduction to Programming

第一组按钮（图2-17 Editor controls）控制着Editor。其中的第一个按钮（一堆横线样子），表示Standard Editor（标准编辑器）。Standard Editor就是新建项目后看到的Editor，Editor里只有一个界面。第二个按钮是Assistant Editor，两个圆圈相交。点击Assistant Editor按钮后，在Standard Editor旁边会出现一个新的Editor，这个新的Editor会呈现出与Standard Editor相关的内容。把Assistant Editor想象成你的管家吧，它会总能提供给你需要的文件。最后一个按钮是是Version Editor，用来追踪和分析你项目的变化。在大部分情况下，使用Standard Editor。



第二组按钮（图2-18 View controls）在Toolbar的最右边，是用来隐藏或者显示Xcode的左中右三大部分。第一个按钮隐藏或者显示Projector Navigator；第二个按钮隐藏或者显示Editor下方的Debugger（在第六章中会详细介绍）；第三个按钮隐藏或者显示Inspector。这三个按钮在你使用小屏幕电脑时会带来很大的帮助。



> Exercise: Hello World | Page 31

接下来点击Project Navigator中的 Main.storyboard。 现在Editor会呈现storyboards（图2-19）。storybaords用来呈现用户界面（user interface）。storyboards包含所有的按钮、图形，以及开发APP所需的元素，同时storyboard也可以适配iPhone到iPad的所有尺寸。



你的第一个APP只运行在iPhone上，所以点击Inspector顶部Toolbar（工具条）上第一个按钮，按钮外表看起来像是折了一个角的纸张。然后到中间部分，不勾选Use Auto Layout选项（图2-10）。接着勾选Disable Size Classes选项。Auto Layout是一组工具，用于在多个屏幕尺寸下自适应，关于这些知识，我们会在第七章中介绍。



> Page 32 | Chapter2: Introduction to Programming

在Inspector的下方，还有一组工具按钮（图2-21），点击其中第三个按钮，那个圆圈中有个方块的按钮，这个按钮展示的是Object Library，就会看到下方的内容发生了变化。



Object Library（图2-22）展示的是常用的用户界面要素。在Object Library中向下滑，找到叫Label的组件，Label是在用户界面中展示文字的，这个文字只能读，用户不能修改Label中展示的文字。用鼠标点击Label然后拖拽到Storyboard中去（图2-23）。





> Exercise: Hello World | Page 33

当你拖动Object Library中的组件到Storyboard中时，Xcode会帮助你垂直对齐和水平对齐，你可能在PowerPoint或者Keynote中见过这种辅助线。把Label放在屏幕的中央，当Label处于水平中央和垂直中央时的位置时，你会看到两条辅助线：一条横线一条竖线。

> Page 34 | Chapter2: Introduction to Programming

接着选中Label，选中Label右上角的小方块然后往右上方拉，在往右上方拉的时候，会出现一个小方块，显示Label的长和宽（图2-25）。把Label拉成至少150pt宽，30pt高，同时重新定位Label，让Label垂直居中水平居中。



> Exercise: Hello World | Page 35

这时再看一下Inspector顶部Toolbar（工具条），这时的Toolbar是有六个图标（图2-26）。每个图标代表了Inspector的一个部分。从左边数第四个图标叫Attributes Inspector，看起来像是箭头朝下的箭，当年把鼠标停留在这个图标上时，会显示出“Show the Attributes Inspector”的提示，点击Attributes Inspector图标，Attributes Inspector会让你修改你在Storyboard中选中的组件的属性（图2-27），你目前选中的是Label，所以你可以在Attributes Inspector中修改Label的属性。



> Page 36 | Chapter2: Introduction to Programming



> Exercise: Hello World | Page 37

点击Label然后观察Attribute Inspector中的选项，有两种方法可以确认Attribute Inspector中展示的属性是Label的属性而不是其他的组件的属性。方法一：如果Label被选中，Label周围会有白色方块；方法二：Attribute Inspector左上角会显示单词Label。

Label有很多你可以改变的属性。Text属性是表示Label对外展示的文字，在Text后面第一行是选择Text的风格：Plain或者Attributed；Text后面第二行是输入框，这里输入的文字将直接在Label上展示。也可以直接双击Label，然后输入“????”，点击回车（图2-28）



Color属性可以改变Label展示文字的颜色，点击Color后面的选项，下拉，找到点击Other。这时出现了一个颜色选择器（Color Picker），你的颜色选择器可能是全黑色的，这时只需将黑色大圆下方的横条右侧按钮滑倒左侧，即可看到彩虹颜色（图2-29）。点击然后拖动大圆中光标，随着位置的变化，Label中展示文字的颜色也会相应变化。选择一个你喜欢的颜色，然后关闭颜色选择器。



> Page 38 | Chapter2: Introduction to Programming

现在我们继续回到Attribute Inspector上，找到font属性，点击后面的T方块，会弹出一个窗口，在Style中选择Bold，然后把Size后面的数字改为20，点击Done按钮（图2-30）。



> Exercise: Hello World | Page 39

在font属性虾米有个alignment属性，点击中间的按钮，让Label展示的文字居中对齐（图2-31），完成这步操作后，你可能需要重新定位Label组件才能让Label垂直居中左右居中。



> Page 40 | Chapter2: Introduction to Programming

接下来我们找到Object Library（图2-32），向下滑找到叫Button的组件，把Button组件拖动到Storyboard中去，把Button组件放在Label组件的上方（图2-33）。





> Exercise: Hello World | Page 41

双击Button组件，然后输入Go。双击能够快速改变文字内容属性，你也可以去属性检查器中修改文字内容。

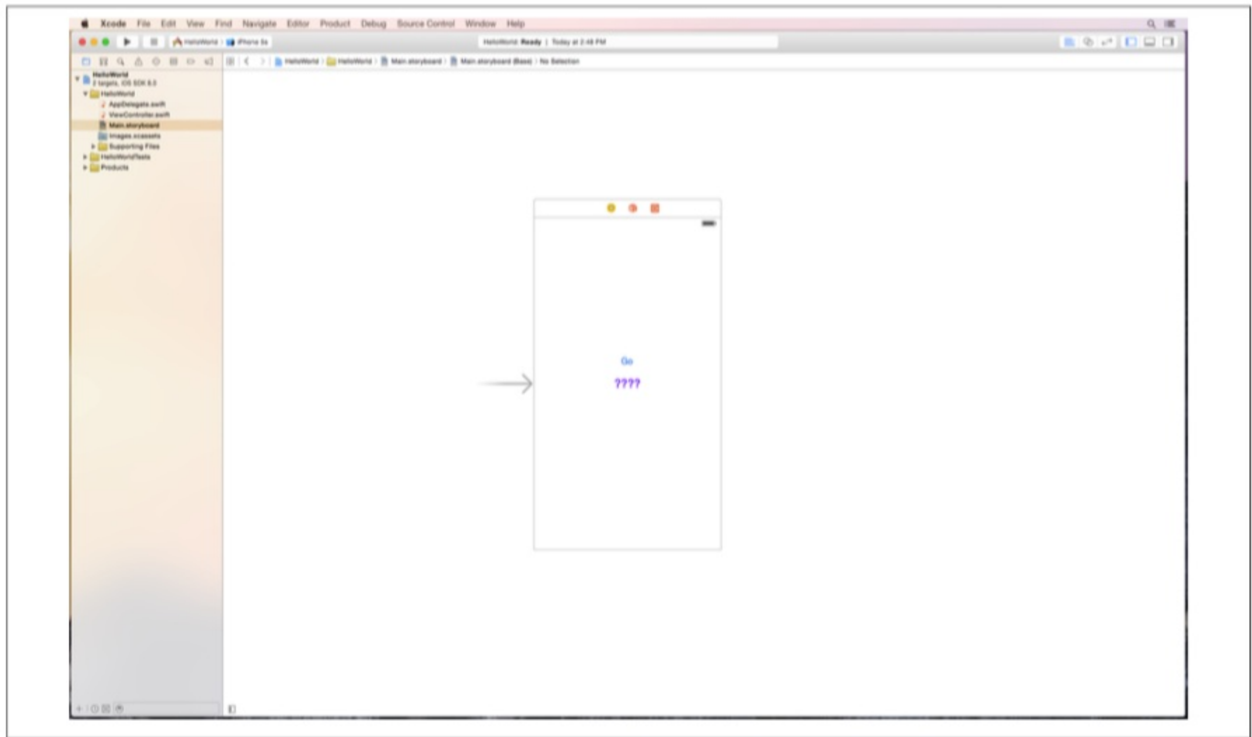
记住，要保存你的工作成果。在Project Navigator的文件中，如果你看到哪个文件看起灰色的暗暗的（图2-34），说明你还没有保存你刚刚进行的改动。点击屏幕最上方左上角第二个File按钮，然后点击Save（图2-35）。当然，你也可以使用Command+s这个快捷方式。



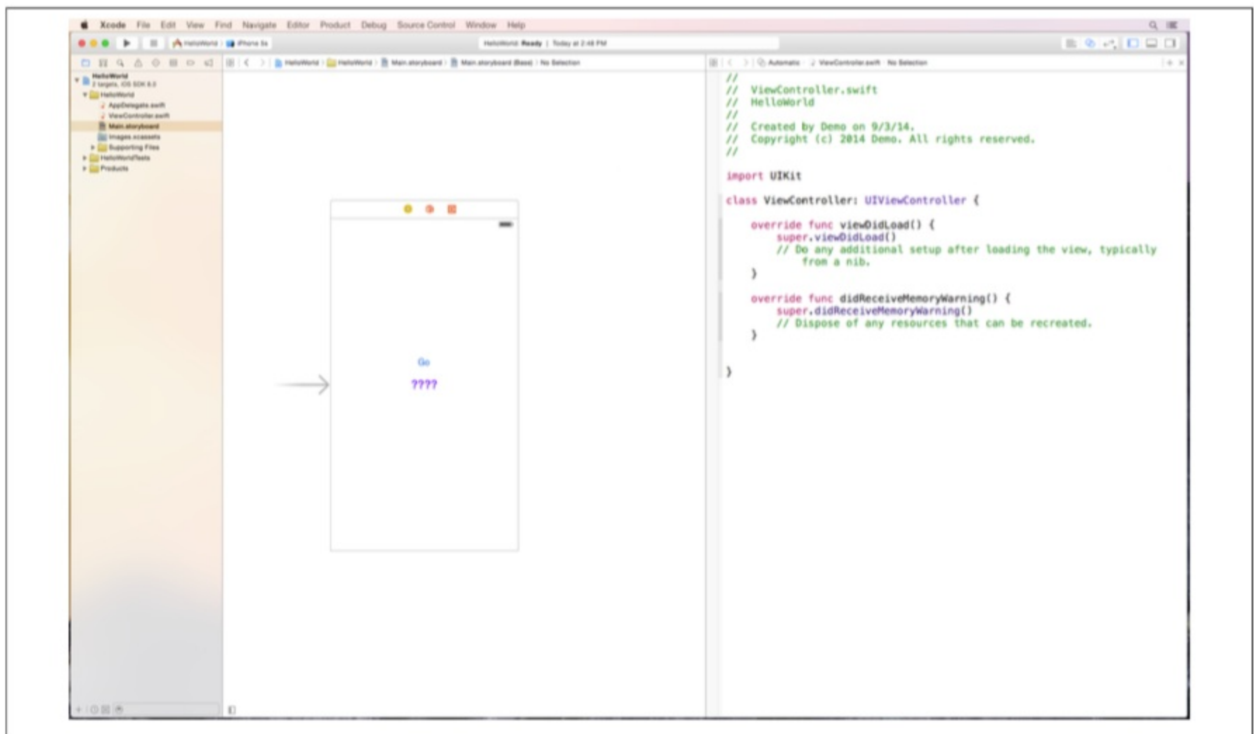


> Page 42 | Chapter2: Introduction to Programming

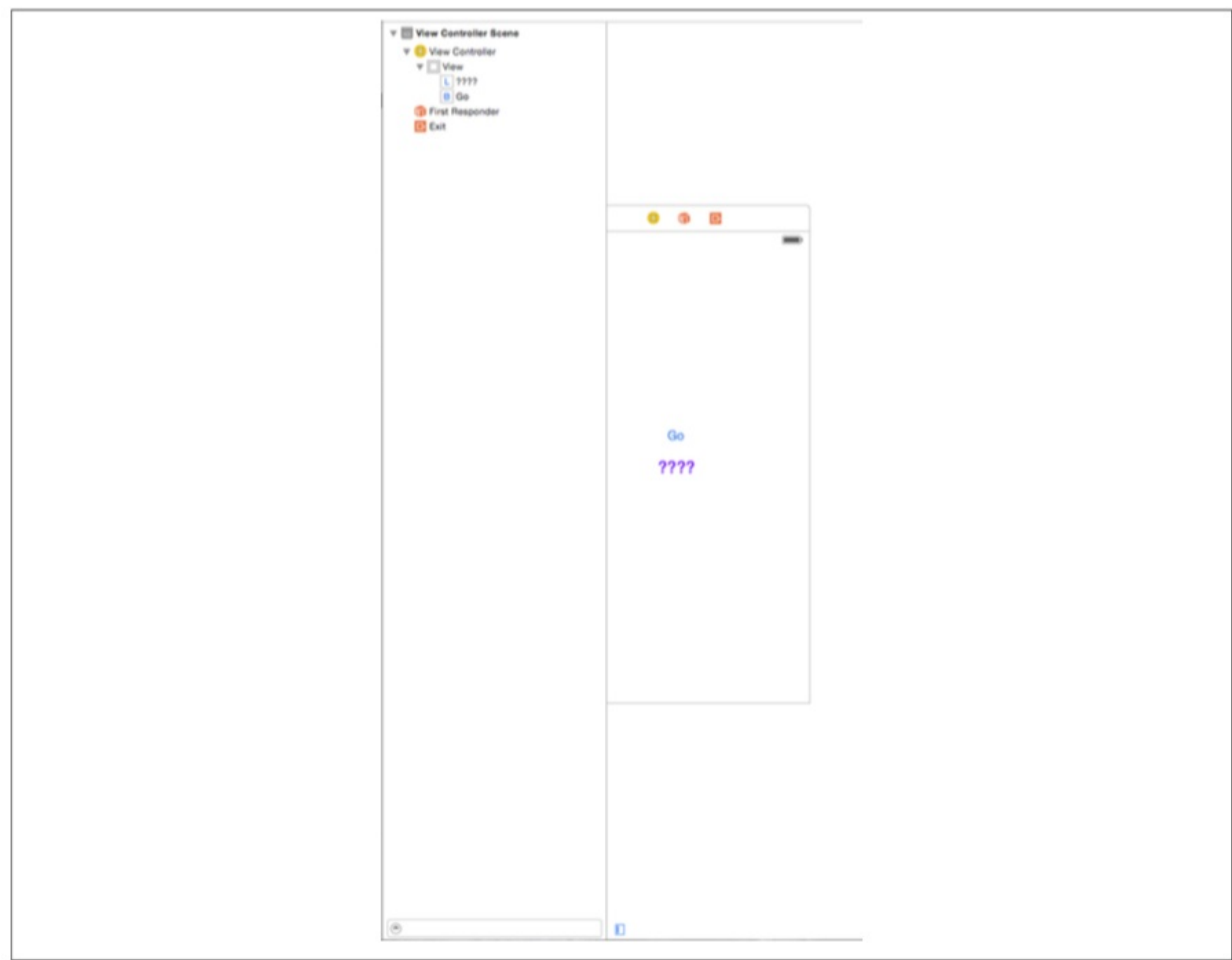
现在你已经在Storyboard中完成了视图的布局，需要写一些代码来控制视图了。第一步，需要把视图中的组件与代码连接起来。先把Inspector隐藏起来，看屏幕的右上角（最上面最右边的按钮），点击一下就可以收起，按钮变成灰色表示收起，蓝色表示展开（见图2-36）。



然后点击右上角两个圆圈相交的按钮，会在右边出现一个新的编辑器Editor，这个编辑器叫Assistant Editor。Assistant Editor会检测到你正在使用的文件，然后找出对应的有关联的文件（见图2-37）。

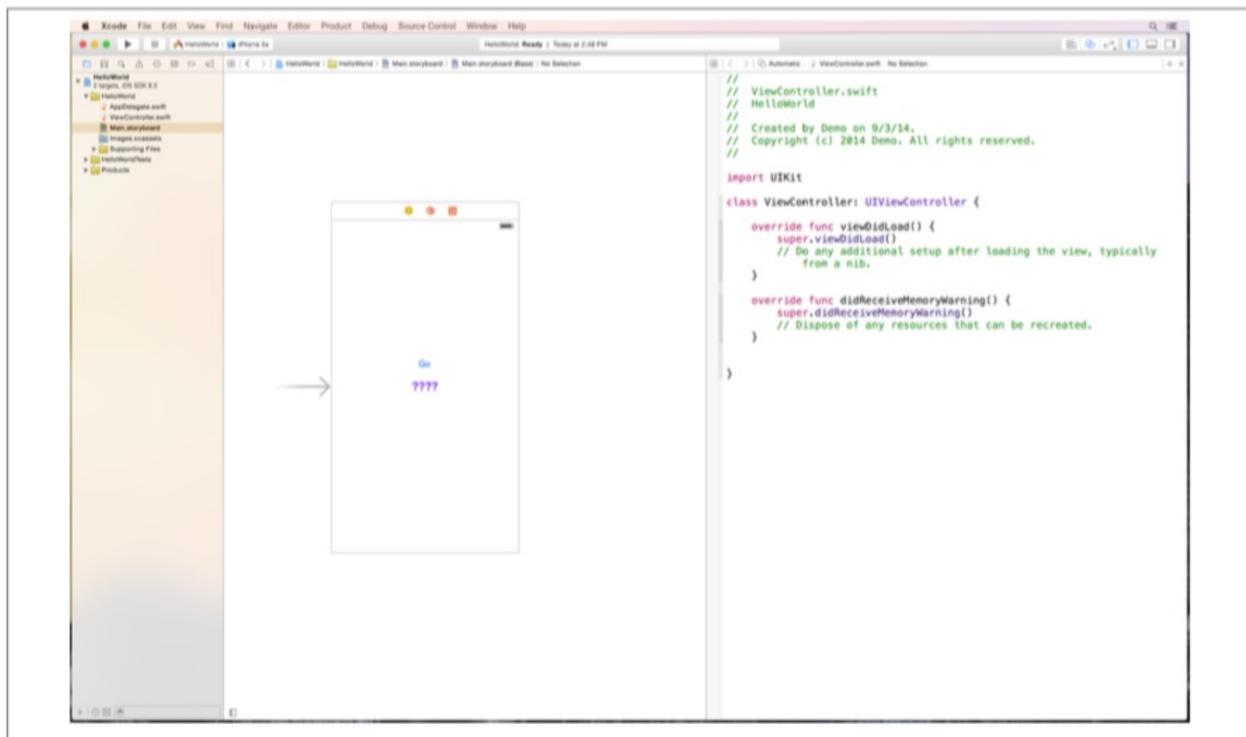


你可能注意到了这时候Stroyborad有部分被一个元素清单遮挡了，这个清单叫做Document Outline，Document Outline可以快速查看选中Storyboard中任何元素（见图2-38）。



Page 44 | Chapter2 : Introduction to Programming

Standard Editor的左下角有一个条纹小正方形按钮，点击一下，可以把Document Outliney隐藏起来（见图2-39）.



Exercise: Hello World | Page 45

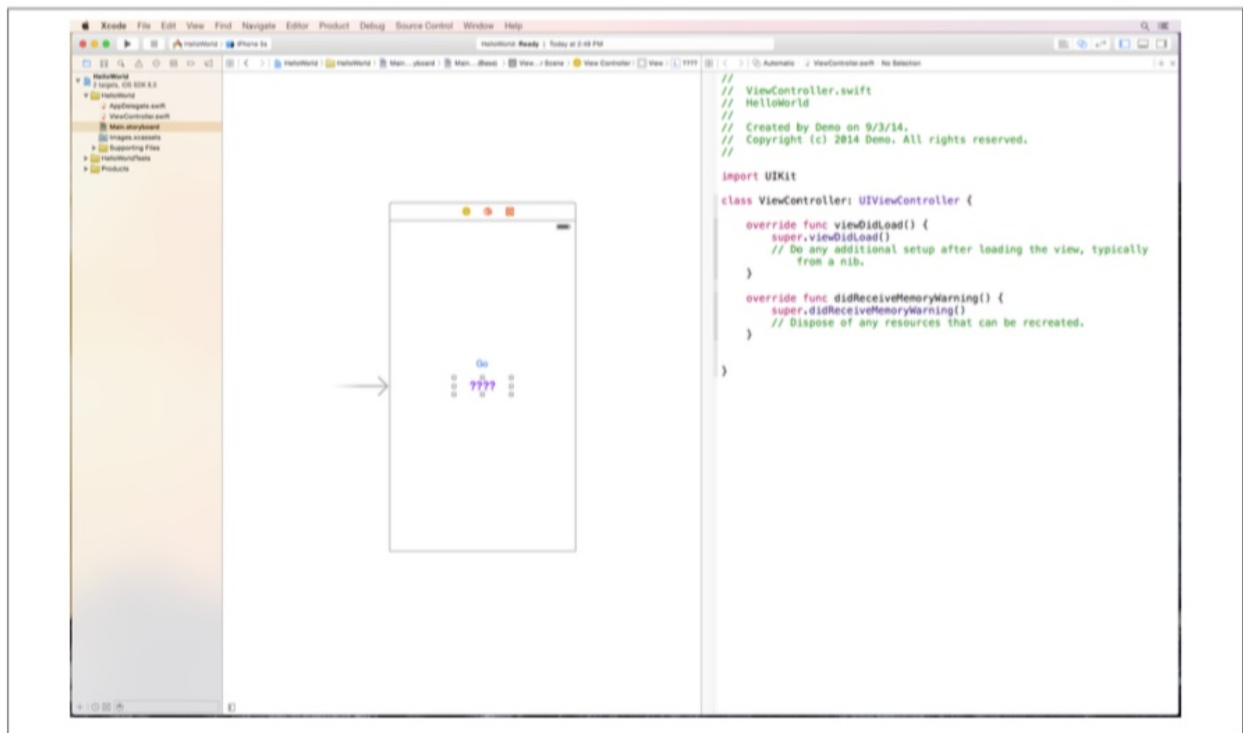
Storyboard

Storyboard是Model-View-Controller模式一个典型代表。当前的Assistant Editor展示的是ViewController.swift文件，ViewController.swift文件是Model-View-Controller中的controller（控制器），控制着你刚刚搭建的界面中的所有是逻辑部分。在把界面中的元素接连到controller（控制器）之前，你需要先理解连接界面（view）和控制器（controller）两种方式的不同点。

第一种连接方式叫做action。当用户在手机界面上点击、滑动、做手势时，就触发了一个action。action然后发送给controller（控制器）一个警告，controller（控制器）决定做出何种反应。一般Button（按钮）这个控件会经常使用action连接。例如，一个Button（按钮）被点击后，controller（控制器）就得到了通知，然后controller（控制器）就决定做出哪种反应。

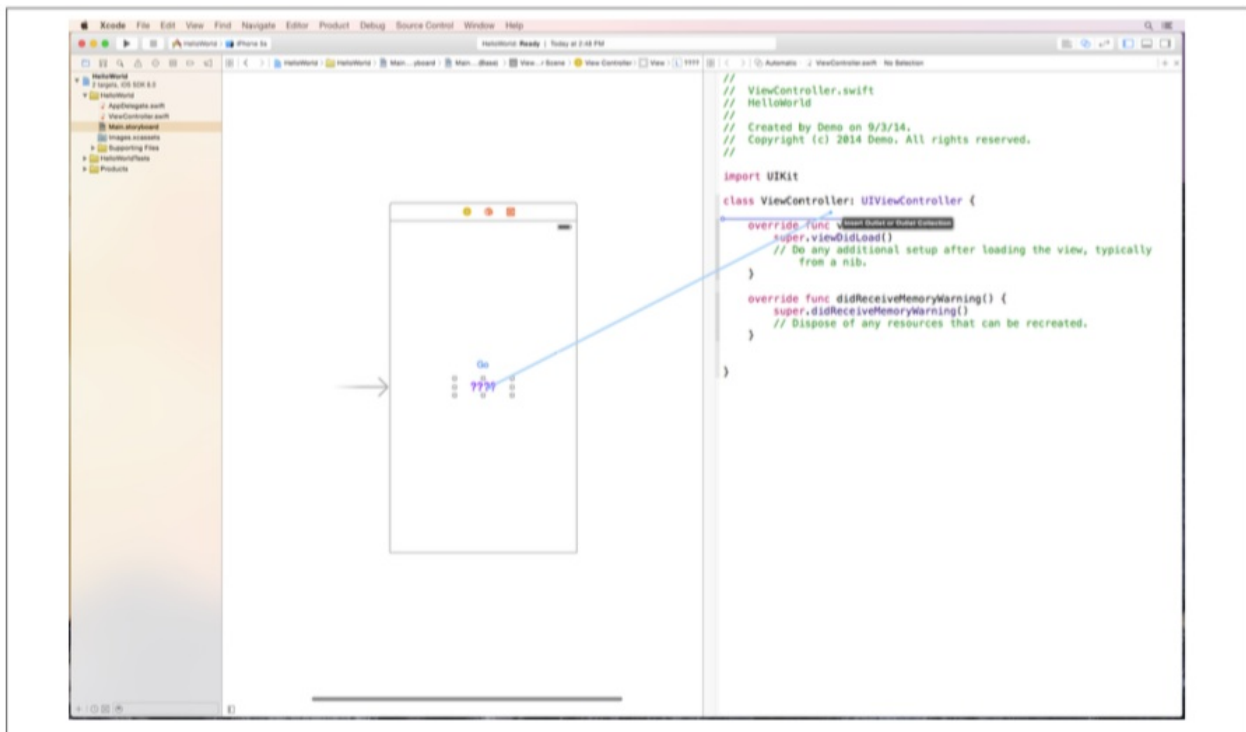
另外一种连接方式叫做outlet。outlet的作用和action截然相反，outlet直接连接view（界面）和controller（控制器），接收或者设置界面控件的信息。例如，一个Label有一个相关的outlet，这样controller（控制器）就可以读取或者设置Label的显示内容。

建立连接的方法就是在view（界面）和controller（控制器）之间拖动，从一方拖动到另一方。在Storyboard中选择一个Label（见图2-40）。



Page 46 | Chapter2 : Introduction to Programming

现在按住Mac键盘上的Control键，注意和Command键区分开来，Controll键在Option键的左边（这是Mac键盘的布局，如果你使用的是PC键盘，估计就不是这个顺序了）。按住Controll键的同时，点击Label然后拖动到Assistant Editor中（见图2-41）。

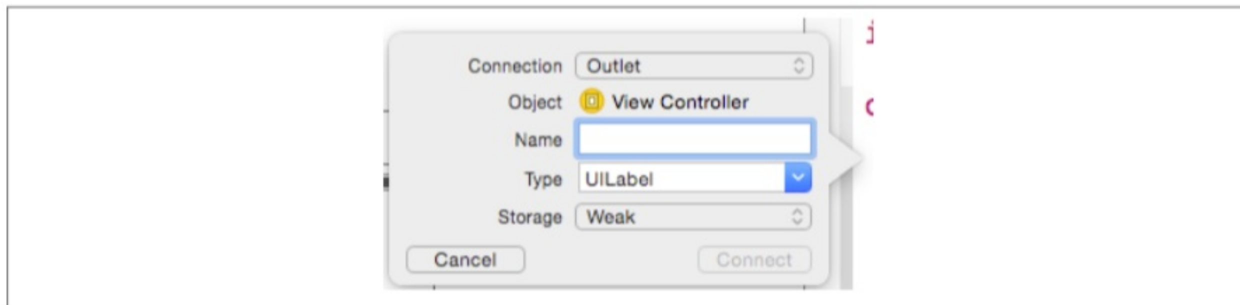


Storyboards | Page 47

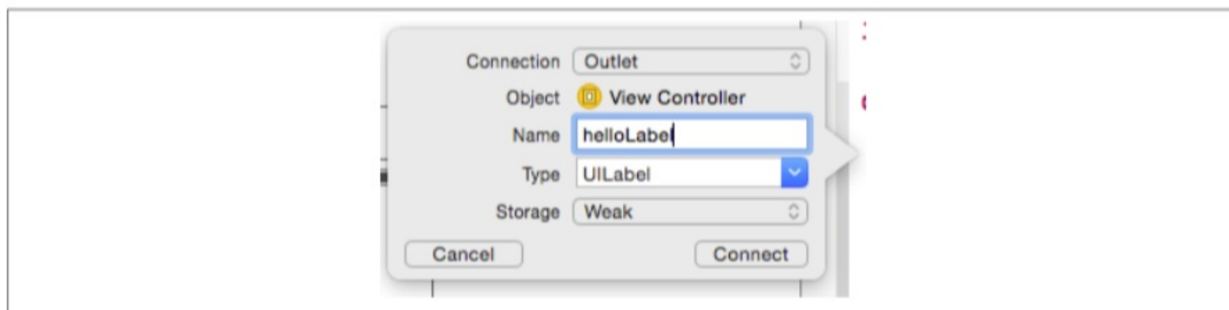
拖动让你的鼠标在下面这行代码的下方：

```
ViewController : UIViewController
```

当你看到深蓝色横线出现在光标下方时，松开鼠标。弹出一个窗口，让你输入一些信息（见图2-42）。



在对话框中，connection后面是灰色无法选择，连接类型只能是Outlet，这是因为Label不能创建action连接。用户无法点击一个Label然后做些什么。我们在Name一栏中输入helloLabel（见图2-43）。



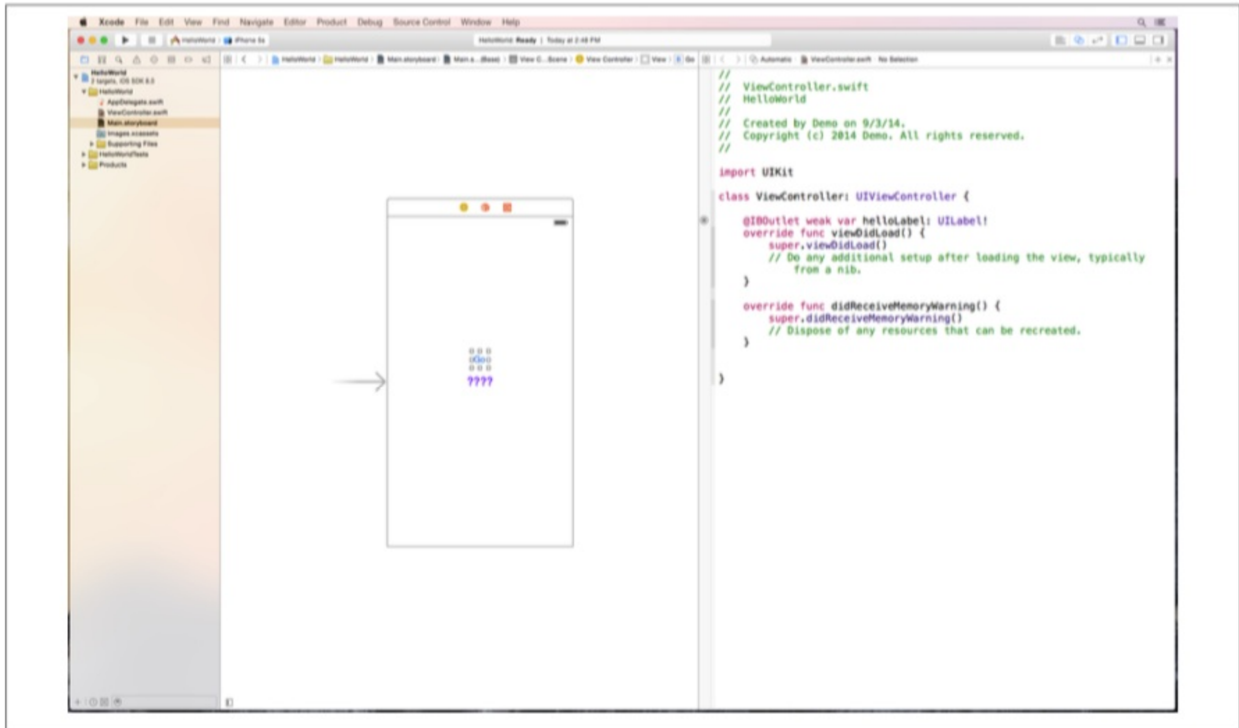
> Page 48 | Chapter2 : Introduction to Programming
>
> 
> 使用驼峰命名法，去掉所有的空格，然后将第二个单词以及往后的单词的首字母大写，第一个单词不大写。例如 number of years就要写成 `numberOfYears`

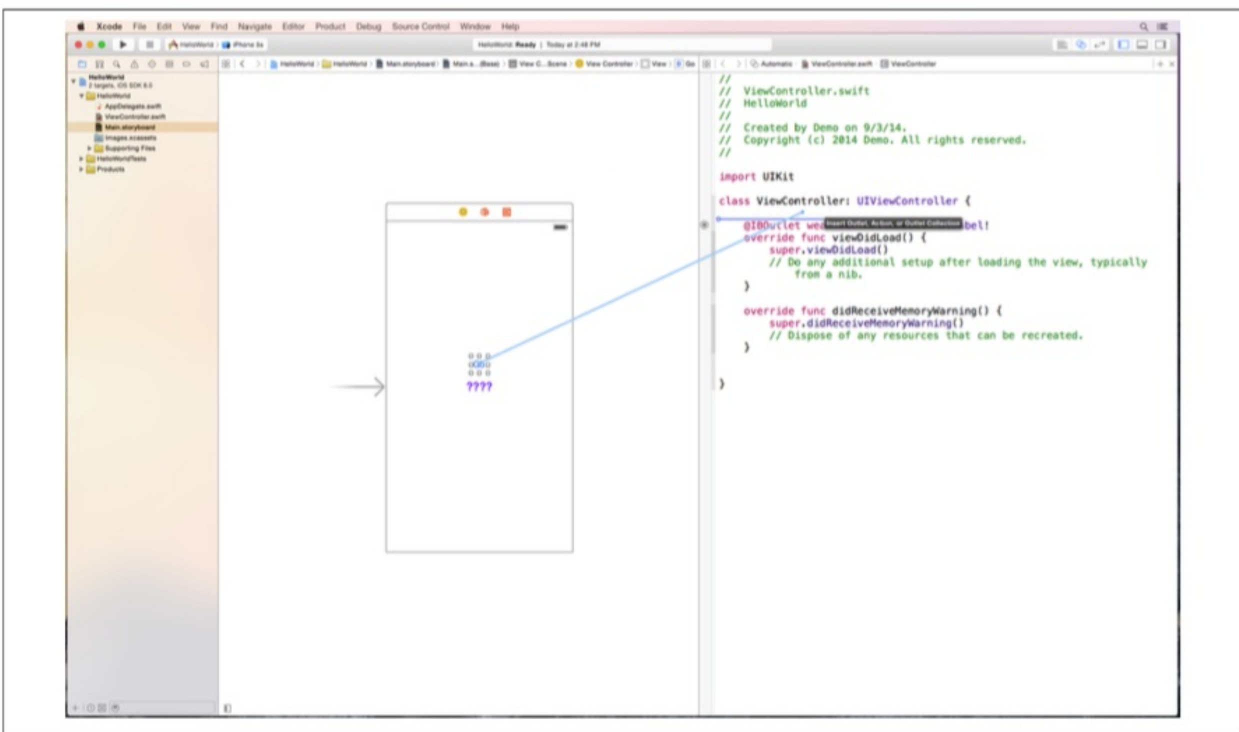
在outlet和action命名时，使用驼峰命名法。

按照最佳实践原则（best practice），最好命名时写上这个控件的类，这样在controller（控制器）中写代码时，比较容易分辨哪个变量对应Storyboard中的哪个控件。输入名字后，点击Connect。

接着点击Storyboard中的Button（见图2-44），然后按住Control键拖动到Assistant Editor中上次相同的位置，看到光标下方出现了一条蓝色横线后，松开鼠标结束拖拽（见图2-45）.

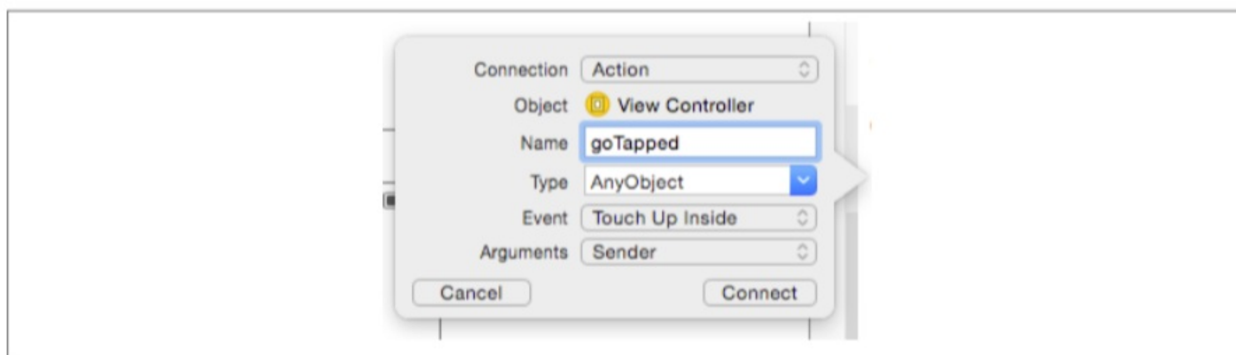
Storyboards | Page 49





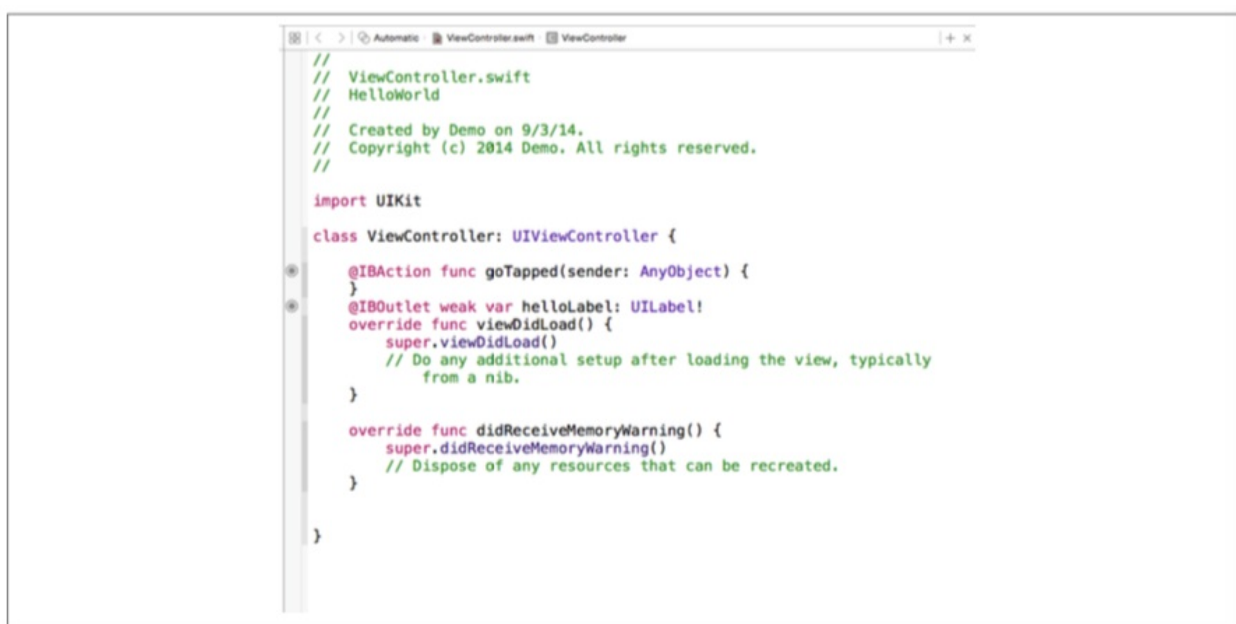
Page 50 | Chapter2 : Introduction to Programming

然后屏幕上弹出连接对话框（a connection pop-up dialog box），把connection后面的类型改为Action，这时事件（Event）选项出现，这个事件（Event）会触发action连接，选择Touch up Inside然后Name一栏中输入goTapped（见图2-46）。



按照最佳实践原则，我们在命名action的时候，往往使用过去式（英文ed结尾）。在这个例子中，Touch Up Inside是指点击相关联的Button然后抬起手指。确保你的连接类型选择的是Action，最后点击Connect。

创建连接后，每一条连接都会在ViewController.swift文件里自动生成代码，以@IBAction或者是@IBOutlet开头（见图2-47）。



> Storyboards | Page 51

把鼠标光标放在下面这行代码的下面：

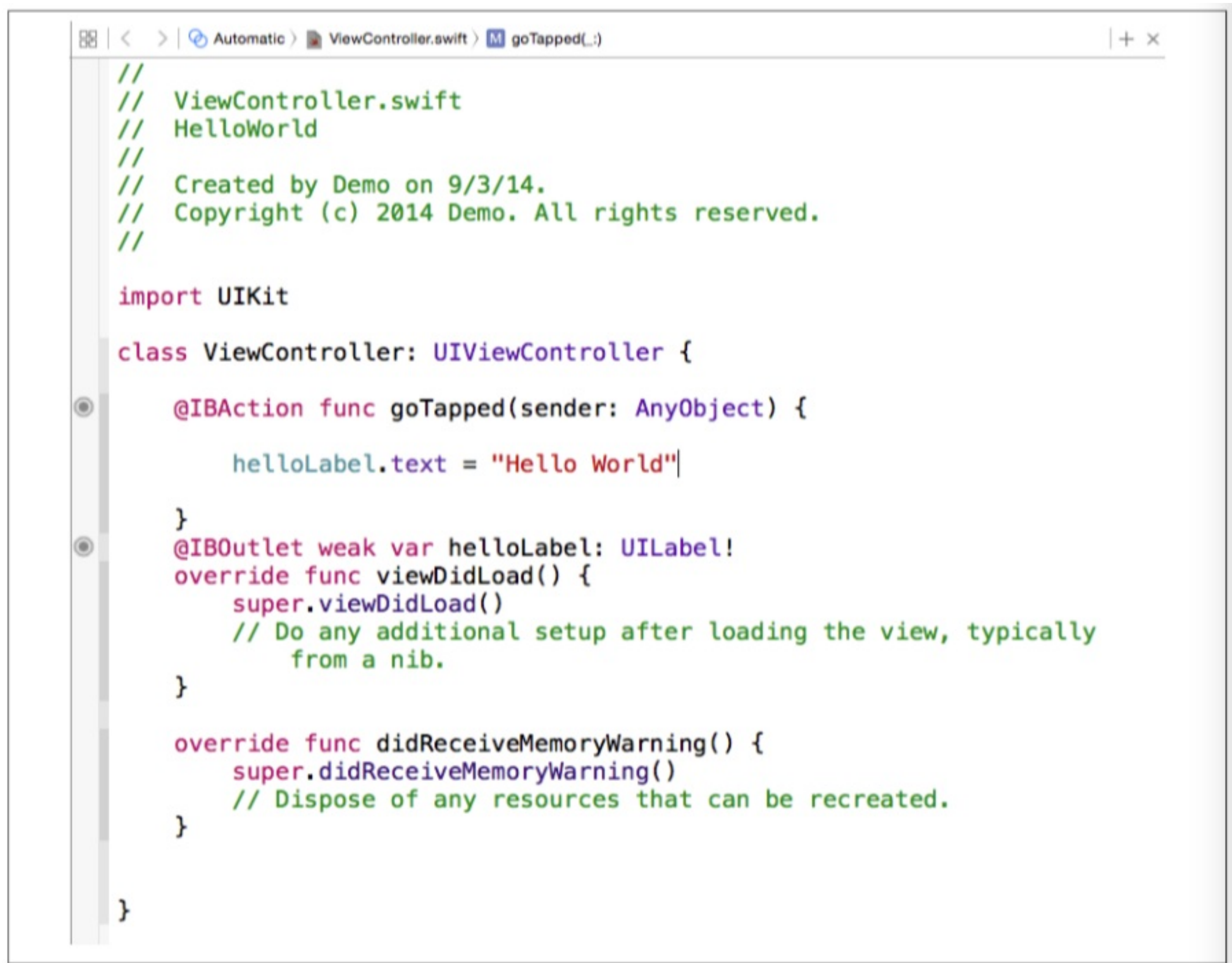
```
@IBAction func goTapped ( sender: AnyObject ) {
```

在两个大括号之间输入下面这行代码：

```
helloLabel.text = "Hello World"
```

这是唯一一行在当前应用中你需要写的代码。变量 `helloLabel` 值是的界面上的Label，在变量后面敲一个点，后面会出现和当前对象相关的所有的方法和属性。在当前这个例子中，出现的是text属性。text属性需要有一个字符串，将字符串内容展示在Label中。等号后面

是 `helloLabel` 的text属性值"Hello World"（图2-48）。



关键时刻到了，点击屏幕左上角的Play按钮，会出现iOS模拟器正在运行你的App。iOS模拟器（iOS Simulator）是模拟iOS系统的虚拟设备，可以模拟所有尺寸的iPhone和iPad。

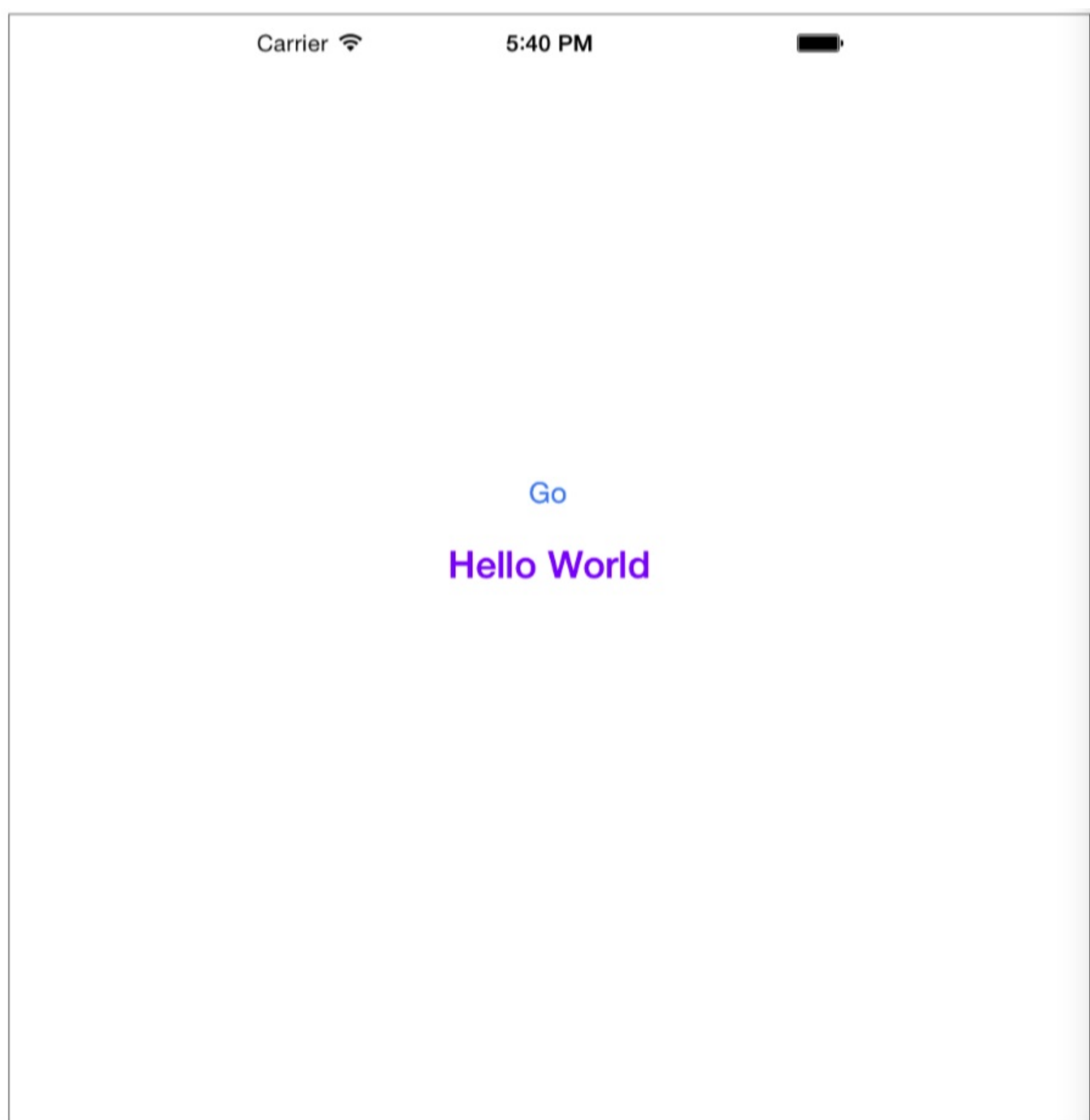
> Page 52 | Chapter2 : Introduction to Programming

如何你想在真机上运行你的App，你需要注册苹果开发者计划。（目前已经不需要了，不需要花钱就可以在真机上运行编写的程序了，只是你想把App上传到应用市场上的时候，才需要花钱购买开发者计划，估计这本书写出来的时候，还没有免费呢）。在第十章中，我们会告诉你如何在真机上运行。

iOS模拟器第一期启动用的时间可能会长一些（见图2-49），之后的运行就会非常快了。如果你没有看到屏幕上出现了一个虚拟的iPhone，点击Dock上的iOS模拟器图标。模拟器运行后，点击Go按钮，然后好好欣赏你的第一个App吧（见图2-50）！



Storyboards | Page 53



如果整个过程都没有遇到问题，那么，恭喜。现在，试着不看书完成所有的步骤，记住一些基本步骤会让你更快的进步。 如果你遇到了问题，报错了，有警告了，也不要担心，犯错是最好的学习方法，熟能生巧，多重来几次。appschool.com/book网站上有一些已经写好的源代码，下载这些代码进行对比，多尝试，一遍一遍的重来，直到你完成App。

> Page 54 | Chapter2 : Introduction to Programming
>
> 这一章就完全结束了，说实话，翻译一章还是比较费时费力的，没有想到翻译是这么不容易。非常开心~

第三章：了解Swift

在这一章里，我们会学习Swift基本的知识：创建变量，集合，循环体以及条件语句。这一章将会介绍Swift中最重要的知识，这样你能更开的开发程序。

什么是Swift？

Swift是苹果公司开发的新的编程语言，在2014年六月的AWDC上推出的。过去20年里苹果公司使用Objective-C来开发程序，而Swift的推出是为了让编程更加简单明了，为初学者减少了障碍，让所有人都可以开发APP。

Playgrounds

Swift最令人幸福的新特性之一就是Playground，可以快速简单的编写程序，能够立即检测代码正确性。我强烈推荐你在阅读此书的时候，电脑上会一直开着一个Playground文件，写下书中的代码和例子，这样不仅能帮你看到代码是如何运行的，更能提供你的记忆力。 现在，新建一个Playground文件吧（图3-1）。



> Page 55

打开Xcode，点击File -> New -> File ，点击iOS下的Source，点击Playground，最后点击Next（图3-2）。



> Page 56 | Chapter3:Diving into Swift
>
> 本书的每一个章节你都要保存一个Playground文件

选择要保存的文件夹，可以保存在上一章节中已经建好的Programming文件夹，输入文件名，点击Create（图3-3）。现在，你有了一个Playground文件了（图3-4）。





> Playgrounds | Page57
>
> 模仿本书的代码，试验你的想法，试验是学习新语言的最好方法。本书的代码可以在这里找到：AppSchool.com/book

声明变量（Creating Variables）

变量是可以改变的值，值是动态的。动态意味着一个值可以现在变化，也可以在将来变化。在swift中，用 `var` 表示变量，这样Xcode就知道你声明的是一个变量。 变量用变量名表示，变量名关联变量的值，这样就意味着，变量的值发生改变后，变量名是不变的，可以一直用变量名来指示这个变量。变量名的命名方法遵循驼峰命名法。例如： `var numberOfYears` 在变量名后面跟一个冒号，这是告诉Xcode你要声明的这个变量的类型。类型就类似电影中的分类，有些是喜剧电影，有些是音乐剧、恐怖片、文艺。每个分类都有自己独特的特性，但是它们都是电影。声明变量的类型使用冒号，在冒号后面，写上类型的关键词。例如，声明一个整型变量，用 `Int` ： `var numberOfYears: Int`

> Page 58 | Chapter3:Diving into Swift

最后，你必须给这个变量设定一个初始值，这个初始值必须符合你设定的类型，否则，这个变量就无法声明： `var numberOfYears: Int = 30` 这里的等号是用了初始化一个变量。初始化表示给变化一个默认值。把这个过程想象成打开一个新玩具，却忘记放电池，这个玩具没法玩直到我们把电池放进去。你必须初始化你的变量，赋一个默认值，不然这个变量无法使用。

整型（Integers）

整型，整数的正值或者负值，用 `Int` 表示。下面这行代码，用 `var` 表示这是一个变量，用 `favoriteNubmer` 作为变量名，紧接着用一个冒号来声明这个变量的类型。这行代码显示，这个变量的类型是整型，所以使用 `Int` 。最后，这个变量的初始值设定为4，跟在等号的后面： `var favoriteNumber: Int = 4`

浮点型（Float）

浮点型是指有小数点位的数值，用 `Float` 表示。例如，下面这行代码，用 `var` 表示这是一个变量，用 `accountBalance` 作为变量名，冒号声明这个变量是 `Float` 类型，最后，用等号表示这个变量的初始值是1203.51： `var accountBalance: Float = 1203.51`

布尔型（Boolean）

布尔型的值，只有 `true` 真或者 `false` 假，两种情况，用 `Bool` 表示。例如，下面这行代码，用 `var` 表示这是一个变量，用 `isStudent` 作为变量名，冒号后声明这个变量是 `Bool` 类型，最后，用等号表示这个变量的初始值是true： `var isStudent: Bool = true`

字符串类型（Strings）

字符串类型是指字符，英文的字母单词或者中文的汉字词汇等字符，要用双引号将字符串包含起来，用 `String` 表示。例如，下面这行代码，用 `var` 表示这是一个变量，用 `firstName` 作为变量名，冒号后声明这个变量是 `String` 类型，最后，用等号表示这个变量的初始值是 Steve： `var firstName: String = "Steve"`

> Creating Variables | Page59
>
> 记住任何时候声明变量时，都要遵循下面这个格式： `var variableName: type = newValueHere`

对象（Objects）

变量也可以表示对象，对象的类型就是类的名字。记住，类就是这个对象的蓝图，类里面包含了对象所有的属性和行为。 下面这行代码，用 `var` 表示这是一个变量，用 `fastCar` 作为变量名，冒号后声明这个变量是对象类型，在这里，就是 `Car` ，最后，用等号表示这个变量的初始值是一个新的车对象： `var fastCar: Car = Car()`

> 个人挑战：在Playground中声明你自己的整型、浮点型、布尔型和字符串类型的变量

常量（Constants）

有时候有些东西你确定它们是百分之百不会改变，比如你的生日。这时候，你可以用常量来代替变量。常量是用来处理那些不会改变的数值，一旦设定值后，在它的整个生命周期内，值都就不会再改变了。常量的值是静态的，静态意味着值在现在或者将来都不会改变。 定义一个常量和定义一个变量非常相似，唯一的区别是定义常量使用 `let` 而定义变量使用 `var` 。例如：

```
var numberOfYears: Int = 30`
let name: String = "Steve"
let isMale: Bool = true
var bankAccountBalance: Float = 1034.20
```

你会注意到name和isMale是常量，这些常量的值是永远不会改变的。你也许会问："为什么我不直接把所有的常量都用变量表示，然后一直不改变变量的值，这个也和常量没有什么区别了啊？"恩，这样也是可行的，只是这个方法违背了编程的最佳实践原则。最佳实践原则能够为开发者和用户提供更好的体验。例如，使用一个常量代替变量能够节省内存和电量，如果所有的开发者都践行最佳实践原则，这样会给用户提供更好的体验。

> 个人挑战：在Playground中给你自己这个人来制作变量和常量，看看你能想到多少变量和常量来描述自己？
>
> Page 60 | Chapter3:Diving into Swift

类型推断（Type Inference）

为每一个变量和常量声明类型可是一件体力活，苹果公司的工程师一直致力于让开发者的工作更轻松。基于你提供的等号右边的值，Swift可以自己判断类型，也就是具有类型推断的能力，这使得声明变量更加容易。请见下面的例子：

```
var numberOfYears = 30
let name = "Steve"
let isMale = true
var bankAccountBalance = 1034.20
```

冒号和冒号后面的类型就不需要了，因为Swift可以根据等号右边的值来推断类型，然后自动设定变量的类型。这样能节省不少时间，也能让代码更简洁。

修改字符串（Modifying Strings）

Swift中的字符串类（String class）提供了不少简便快捷的方法，例如 `uppercaseString` ，可以使小写字母改变为大写字母。//后面展示的是改变后的字符串效果。

```
var favoriteFood: String = "Pasta"
favoriteFood = favoriteFood.uppercaseString    //PASTA
```

当你设定favoriteFood变成favoriteFood.uppercaseString时，等号表明等号右边的值赋值给等号左边的变量，把这个过程拆解开来就是：

```
 1. favoriteFood  //"Pasta"
    2. favoriteFood.uppercaseString  //"PASTA"
    3. favoriteFood = favoriteFood.uppercaseString   //"PASTA"
    4. favoriteFood   //"PASTA"
```

> Modifying Strings | Page61

大部分的代码会按顺序执行，第一行代码运行完毕后，编译器会运行第二行代码，接着第三行代码，依次进行。第一行代码在没有执行完毕之前，第二行代码是不会执行的。

>  个人挑战：如果你把上面的第三行从Playground中去掉，会发生什么？原因是？

当然，也有lowercaseString方法，把所有的字母都变成小写字母，还有capitalizedString方法，把所有的单词的首字母大写。

附加字符串（Appending Strings）

你可以在一个字符串变量的后面加上一串字符，也就是附加（appending）：

```
var beach = "Beach"
beach = beach + "Ball"       //"BeachBall"
```

一个字符串可以和另一个字符串相加，组成一个新的字符串。例如：

```
let firstName = "Steve"
var lastName = "Derico"
lastName = firstName + lastName    //"SteveDerico"
```

还可以，在两个字符串之间附加一个字符串。例如：

```
let firstName = "Steve"
let lastName = "Derico"
var fullName = firstName + " " + lastName
var favoriteMovie = " Back to The Future"
var movieString = " My name is " + fullName + "  and my favorite movie is " + favoriteMov
```

字符串中的变量（Variables in Strings）

使用字符串插入（string interpolation），可以直接在变量中增加字符串。字符串插入这个术语是指在一个字符串里使用占位符来产生一个字符串。程序运行后，这些占位符就会直接调用。在这里，字符串是用双引号表示的，那么占位符是用 `\()` 表示的，两个括号中是需要插入的字符串。字符串插入这个方法可以将一个非字符串的变量方便地转换为字符串类型：

```
let seatsPerRow = 25
let numberOfRows = 25
var seatsString = "In the theater, there are \(numberOfRows) rows and \(seatsPerRow) seat
```

Page 62 | Chapter3:Diving into Swift

集合（Collection）

在某些情况下，有必要将许多变量或常量更好地组织在一起，Swift提供2种集合类型来保存和组织这些变量。

数组（Arrays）

假设你想存储过山车中站在排的人的名字，你该怎么做？你可以为在一排中的每一个人声明一个变量，不过这样就会有些麻烦，你也不知道在某个时刻有多少人是在一排上。为了解决这种麻烦，我们用到了数组。 数组（Array）是按照聚体顺序存储多个变量的容器。数组可以存储几乎无限的元素（item），同时每个元素都有一个数组下标，准确标明此元素在数组中的位置。可以这样声明一个数组：

```
var names: [ String ] = [ "Steve", "Jeff", "Andy", "Andrew", "Cole", "Mike", "Mikey" ]
```

开头还是 `var`，接着是冒号，然后是方括号，方括号里是数组的类型，等号的右边，用方括号括起来所有的数组元素，里面每个数组元素用逗号分开。 在Swift中，数组中所有的元素必须是同样的类型，这以为着一个数组能存储所有的字符串，如上面的例子，但是不能存储整

型和字符串2种不同的类型的元素。数组只能存储同样类型的变量。 对于一个既定的数组，Swift可以自行判断出类型，没有必要专门写出数组的类型，所以上面的数组也可以写成这样：

```
var names = [ "Steve", "Jeff", "Andy" ]
```

> Collections | Page63

查找数组（Navigating arrays）

（Navigating arrays是查找数组的意思吗？导航数组？） 数组中的每个元素都有特定的位置或者下标，下标是从0开始的，第一个元素的下标是0，第二个元素的下标是1，第三个元素的下标是2，依此类推。获取数组中的某个元素使用下面的格式：

```
names[0]    // Steve
names[1]    //Jeff
names[2]    //"Andy
```

一开始可能不习惯零开头，时间长了，就会下意识的认为是以零开头了。下标都是以零开头的。

> 
> 个人挑战：声明一个数组，来记录你的家庭成员

使用 `count` 方法可以获得数组中所有元素的总数。最后一个元素的下标会比总数少了一：

```
names.count         // 3
```

检查数组元素是否为空，这个数组中是否有元素，使用 `isEmpty` 方法：

```
var cats = [ "Big Kitty" ]
cats.isEmpty        //false
```

修改数组（Modifying arrays）

用var声明的数组是动态的可以按照需要改变，在数组最后一个元素后面附加一个元素使用append方法：

```
var names = [ "Steve", "Jeff", "Andy" ]
names.append ( "Wally" )         //  [ "Steve", "Jeff", "Andy", "Wally" ]
```

也可以给你的数组增加一个数组：

> Page 64 | Chapter3:Diving into Swift

```
var names = [ "Steve", "Jeff", "Andy", "Wally" ]
var parents = [ "Mike", "Adam", "Nick" ]
names = names + parents           // [ "Steve", "Jeff", "Andy", "Wally", "Mike", "Adam",
```

可以通过某个元素的下标来替换元素值，给这个元素赋新的值： `names[6] = "Sam"` 还有一个 `insert` 方法，可以在某个具体位置插入元素。插入元素后，此元素后面的元素的下标值都会自动增加1。

```
var names =  [ "Steve", "Jeff", "Andy", "Wally", "Mike", "Adam", "Sam" ]
names.insert ( "Buster", atIndex: 2 )      // [ "Steve", "Jeff", "Buster", "Andy", "Wally
```

而删减某个元素，使用 `removeAtIndex` 方法：

```
var names = [ "Steve", "Jeff", "Buster", "Andy", "Wally", "Mike", "Adam", "Sam" ]
names.removeAtIndex(2)       // [ "Steve", "Jeff", "Andy", "Wally", "Mike", "Adam", "Sam"
```

词典（Dictionaries）

数组不是唯一的集合类型，词典也可以存储多个变量，用键（Key）和值（value）将多个变量组织在一起。键值的工作原理和你书架上的牛津大词典类似，键（Key）是你要查找的单词，而值（value）是这个单词的释义。词典是无序存储的，所以只能使用键（key）来获取某个值（value），例如：

`var homeruns : [ String : Int ] = [ "Posey" : 24, "Pagan" : 19, "Pence" : 15 ]` 在这个例子中，有三个键（keys）： `"Posey"` ， `"Pagan"` ， `"Pence"` ,每个键都有对应的值。提高相关的键，写在中括号中，就能获取相对应的值： `homeruns[ "Posey" ]` // 24 增加一对键值：

```
var homeruns : [ String : Int ] = [ "Posey" : 24, "Pagan" : 19, "Pence" : 15 ]
homeruns[ "Sandoval" ] = 10    // [ "Posey" : 24, "Pagan" : 19, "Pence" : 15, "Sandoval"
```

> Collections | Page65

给键设定一个新值就可以改变词典中的元素：

```
homeruns [ "Posey" ] = 36        //  [ "Posey" : 36, "Pagan" : 19, "Pence" : 15, "Sandova
```

将键对应的值设置为空（nil），就可以删除这对键值。nil是空值，我们将在章节结束的时候更深入的介绍nil。删除键值方法：

```
homeruns [ "Sandoval" ] = nil     //   [ "Posey" : 36, "Pagan" : 19, "Pence" : 15 ]
```

循环（Loops）

想象一下你在健身馆运动，你的健身教练告诉你要举重8次，同样的举重动作你要进行8次。编程中处理重复工作使用循环（Loops）。循环用来重复某段特定代码。

For条件递增语句（For-Condition-Increment）

最常见的循环就是For条件递增语句，简称for loop（for循环体）。这个循环有三部分构成：一个变量，一个条件，一个递增。这里的变量常常使用整型，来跟踪目前的已经完成的循环次数。这里的条件会在每次循环结束之后都会检查一次，如果条件结果为真，循环继续，如果条件为假，循环停止。最后，这里的递增是每次循环执行后需要增加的数值：

```
 for ( var counter = 0; counter < 8; counter++ )  {
        liftWeights( )
 }
```

语法是这样的：用 `for` 作为循环的开始，告诉Xcode你要声明一个循环了，for后面跟着括号，货号里面声明变量、条件和递增数值。例如：

```
for ( VARIABLE; CONDITION; INCREMENT )  {

}
```

括号中的第一个部分是变量，用counter表示，计算已经完成的循环的数量，在平时编写程序时，这里的变量常常命名为counter（英文中counter有计数器的含义）然后设定初始值为零：

```
for ( var counter = 0; CONDITION; INCREMENT )  {

}
```

> Page 66 | Chapter3:Diving into Swift

接下来，在变量后面使用分号和条件分隔开来。在这个例子中，条件设定为counter小于8，这将会使循环执行8次。例如：

```
for ( var counter = 0; counter < 8; INCREMENT )  {

}
```

条件后面的一个分号后面是递增值，递增值就是每次循环后变量counter的变化：

```
for ( var counter = 0; counter < 8; counter++ )  {

}
```

大部分的递增值用两个加号这种简写方式。平时要给一个变量加1，语法一般是这样的：`counter = counter + 1` 这行代码表示变量counter加上1的值覆盖counter作为counter的新值，在执行for循环时，counter每次加1。然而苹果公司提供了简写方式，用两个加号 `++` 表示这个变量加1，例如： `counter++` 作用和这个相同： `counter = counter +1`

for-in循环

每次写循环语句都要写变量、条件和增值，这可是一件费时费力的体力活。而for-in循环提供了一种更简单的方法去遍历（循环访问）一个数组或者词典。for-in循环可以自动的检测条件和递增值。此外，还会提供一个变量来表示目前数组中的元素。例如：

```
var favoriteStates = ["California", "New York", "Colorado", "Oregon"]
  for state in favoriteStates {
      println ( "\(state) is one of my favorite states" )
 }
//California is one of my favorite states
//New York is one of my favorite states
//Colorado is one of my favorite states
//Oregon is one of my favorite states
```

`println` 方法，发音为"print line"，能够生成字符串内容然后将内容展示在Debugger中（也就是常说的“打印”）。Debugger会在第六章中介绍。

> Loops | Page67

for-in循环同样也可以在词典中使用，遍历词典中所有的键值，不过需要为键值设置2个变量：

```
var homeruns = [ "Posey" : 24, "Pagan" : 19, "Pence" : 15 ]
for (name, hrs) in homeruns {
     println( " \(name) has \(hrs) Homeruns this season.")
}
//Posey has 24 Homeruns this season.
//Pagan has 19 Homeruns this season.
//Pence has 15 Homeruns this season.
```

区间（Ranges）

区间Range和整型数组中一个数字到另一个数字差不多，目前有两种区间，一种是闭区间，用三个小点表示，包含小点两边的数字：

```
1...5   //1, 2, 3, 4, 5
```

另外一种是半闭半开区间，用两个小点加上一个小于号表示，小于号右边的数字不包含在这个区间中：

```
1..<5     //1, 2, 3, 4
```

在for-in循环中，可以使用区间来代替数组或者词典：

```
for index in 1...5 {
    println ("The current number is \(index)")
}
 //The current number is 1
 //The current number is 2
 //The current number is 3
 //The current number is 4
 //The current number is 5
```

条件表达式（Conditional Statements）

你是否在餐馆遇到过会给过生日的人唱歌的情景？这就是一个条件表达式，在程序中，条件表达式是用来做决定的。

> Page 68 | Chapter3:Diving into Swift

if表达式（if Statements）

类似这种做决定的情况或者可以判断是和否的逻辑问题，都可以使用条件表达式。if表达式决定一个情况为真或者为假，如果条件为真，某段代码就会执行，如果为假，则这段代码不会执行。例如：

```
if isBirthdayToday == true {
    singBirthdaySong ( )
}
```

条件表达式以if开头，然后跟着条件，在上面的这个例子中，条件是 `isBirthdayToday == true`， 两个等号表示比较2个等号之间的数值，如果值相同，则结果为真，如果值不相同，则结果为假。如果为真，就会执行 `singBirthdaySong ( )` 这个方法，如果为假，就会不执行 `singBirthdaySong ( )` 这个方法。最后，两个大括号表示这段代码的开始和结尾。

if-else表达式

在歌剧结束后，按惯例演员会出来谢幕，性别不同会有不同的动作，要么鞠躬，要么行屈膝礼。如果你想在Swift中表达这件事，可以这样写：

```
if isMale == true {
    bow()
} else {
    curtsy()
}
```

在这里例子中，我们需要看isMale是否为真，如果为真，`bow()` 将会执行，如果为假，`curtsy()` 将会执行，上面和下面两部分是相互排斥的，换句话说，你只能执行其中一个，要么上面，要么下面。 有时候你需要检查多个情况才能决定下一步做什么。比如你早上起床后，在工作日会有一套例行事宜，而在周末就会有另外一套例行事宜：

```
if isWeekday == true {
    getReadyForWork()
} else if isSaturday == true {
    goRunning()
} else {
    goToYoga()
}
```

> Conditional Statements | Page69

在这个例子中，你首先要检查今天是不是工作日，如果今天是工作日，就要执行 `getReadyForWork()` ，然后跳过if语句中剩下的代码。如果今天不是工作日，就要跳过第一部分然后进入else if条件。 如果第一个条件为假，else if会被调用。如果今天是周六，`goRunning()` 代码执行，然后跳过剩下的代码。 最后，如果上面两个条件都为假，那么else就会被调用。如果今天既不是工作日也不是周六，那么 `goToYoga()` 会执行。 在日常的编程中，if条件句是非常有力的工具，经常使用if来控制逻辑。记住，要检查每个条件，确保结果只有真或假两种情况。

可选类型（Optionals）

可选值是用来处理那些可能出现空值的变量。在某些情况下，你是无法确保一个变量是不是一定有值。例如，在西班牙语中的一个单词，可能无法直接翻译成英语的一个单词，这样就会出现空值。这种没有值的情况叫做nil。 可选值可以用在任何类型的变量中，在使用时将一个问号跟在类型后面，表示这是可选值： `var translatedWord: String?` 因为可能为空的变量都必须名称表示，这样能确保所有的非可选值变量都会有值。这种设计模式帮助开发者避免了空值引起的程序崩溃。非可选值变量都必须有值，可选值变量可以没有值。 可选值不能直接使用，在使用之前需要解包（unwrapped）。把使用可选值变量想象成拆开一袋糖果，必须先要把包装撕掉才能吃到糖果。当一个可选值变量解包后，这个变量也可能是空值。这就

相当于你拆开一颗糖果，结果发现里面什么也没有。 解包的过程帮助开发者记住去检查然后确保这个变量不是空值，用可选值有2个步骤，第一步，检查是不是为空，一般情况下用if表达式检查：

```
var translatedWord: String? = translate("cat")
if translatedWord != nil {
       //translatedWord has a value
 } else {
     //The translatedWord has no value
 }
```

> Page 70 | Chapter3:Diving into Swift

一旦核查确实有值后，你必须解包。解包一个可选值非常简单，直接放一个叹号在变量后面即可，例如：

```
var translatedWord: String? = translate("cat")
if translatedWord != nil {
    println(translatedWord!)    //gato
}
```

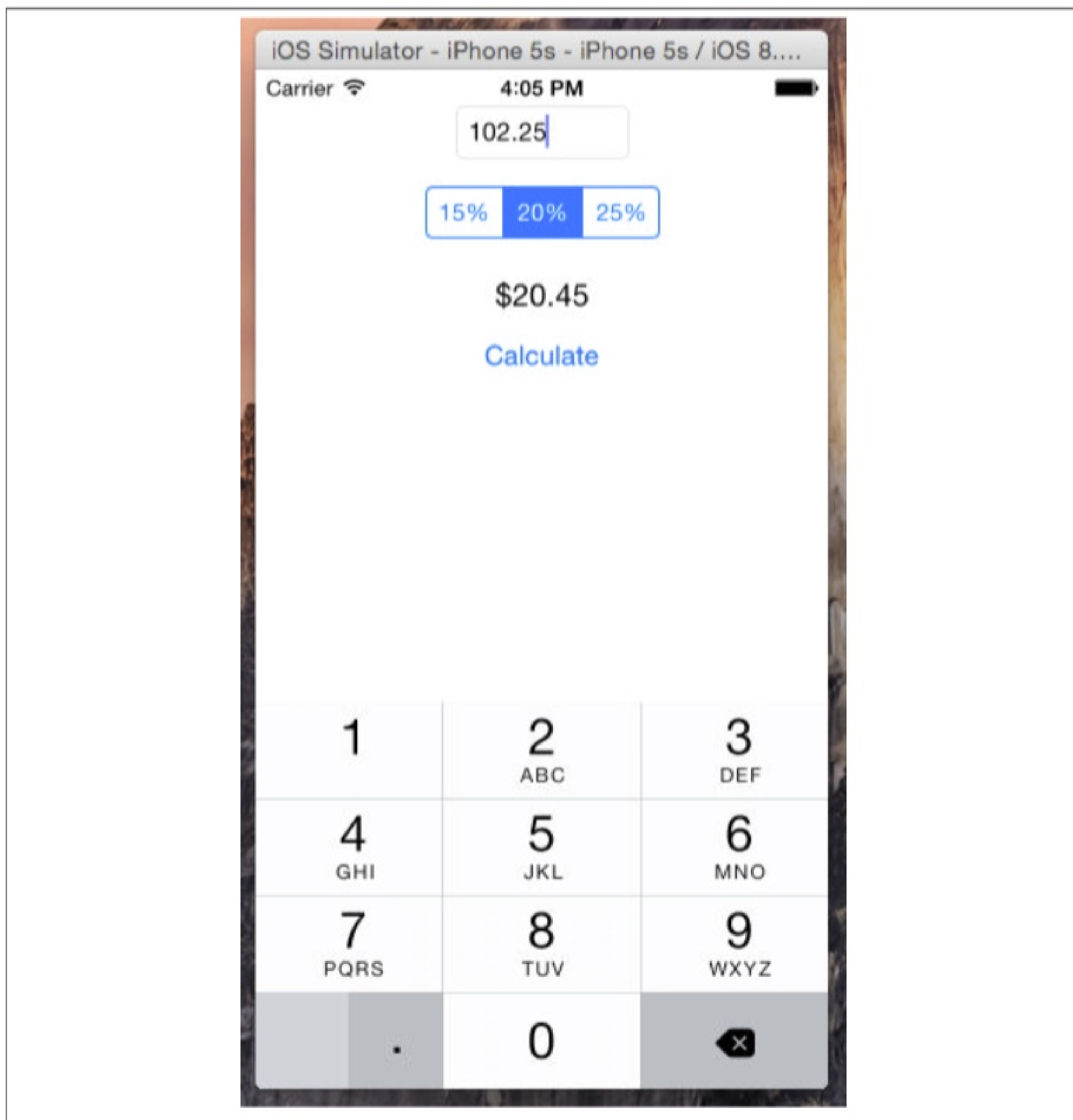
刚刚开始接触可选值的时候会有一些困惑和不习惯，其实你只要记住，一个可能为空的变量必须是可选值，而当可选值为空时就叫做nil。

在这一章节，你学会了Swift的基本知识，你现在能够声明变量、常量、甚至是可选值了。你熟悉了如何声明字符串，整型，浮点型，数组和词典，你也理解了如果声明一个循环体，一个条件语句。现在，是时候让你接受一下检测了，再接再励完成小费计算器程序吧~

第三章练习Tip Calculator - 编写一个小费计算器App吧

练习：Tip Calculator 小费计算器

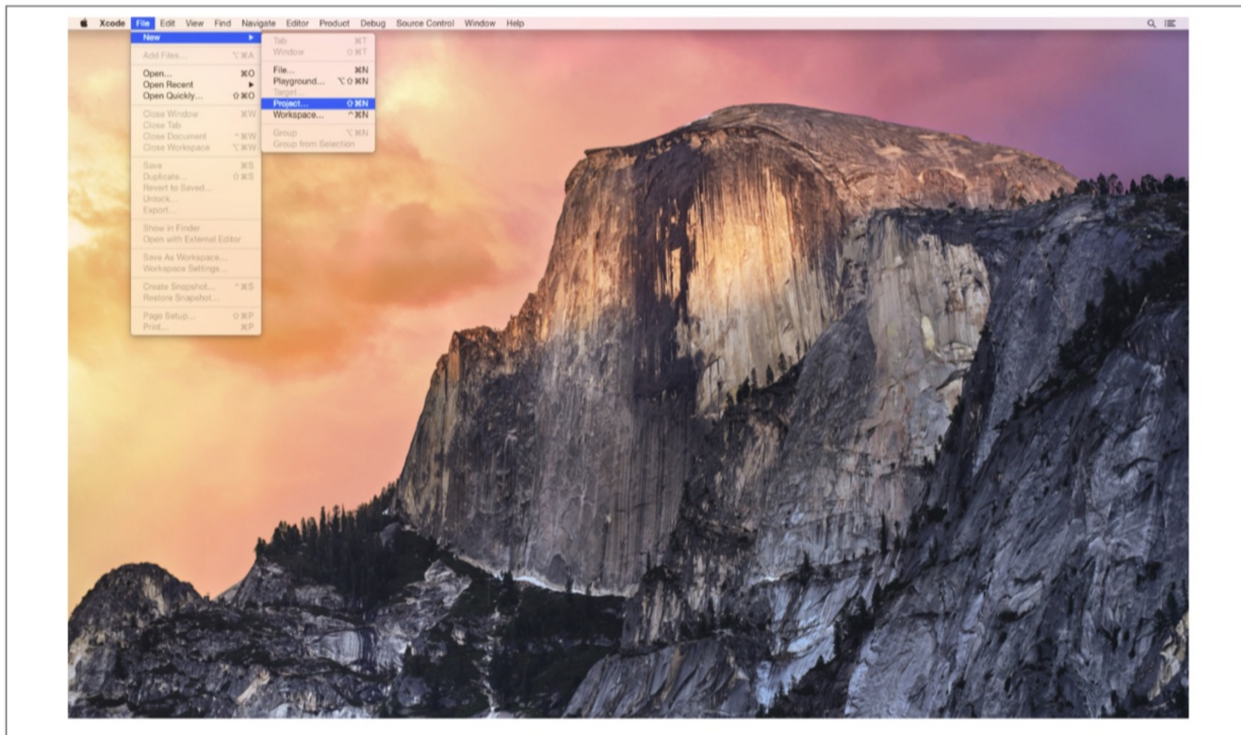
下一个App可以帮助用户计算他们每次去餐厅吃饭时应该付多少钱给服务员吗，用户输入总消费额，然后从3个费率：15%，20%和25%中挑一个费率，用户点击计算按钮，消费数额就会出现在屏幕上。App完成版本的效果图如下（见图3-5）：



Exercise: Tip Calculator | Page 71

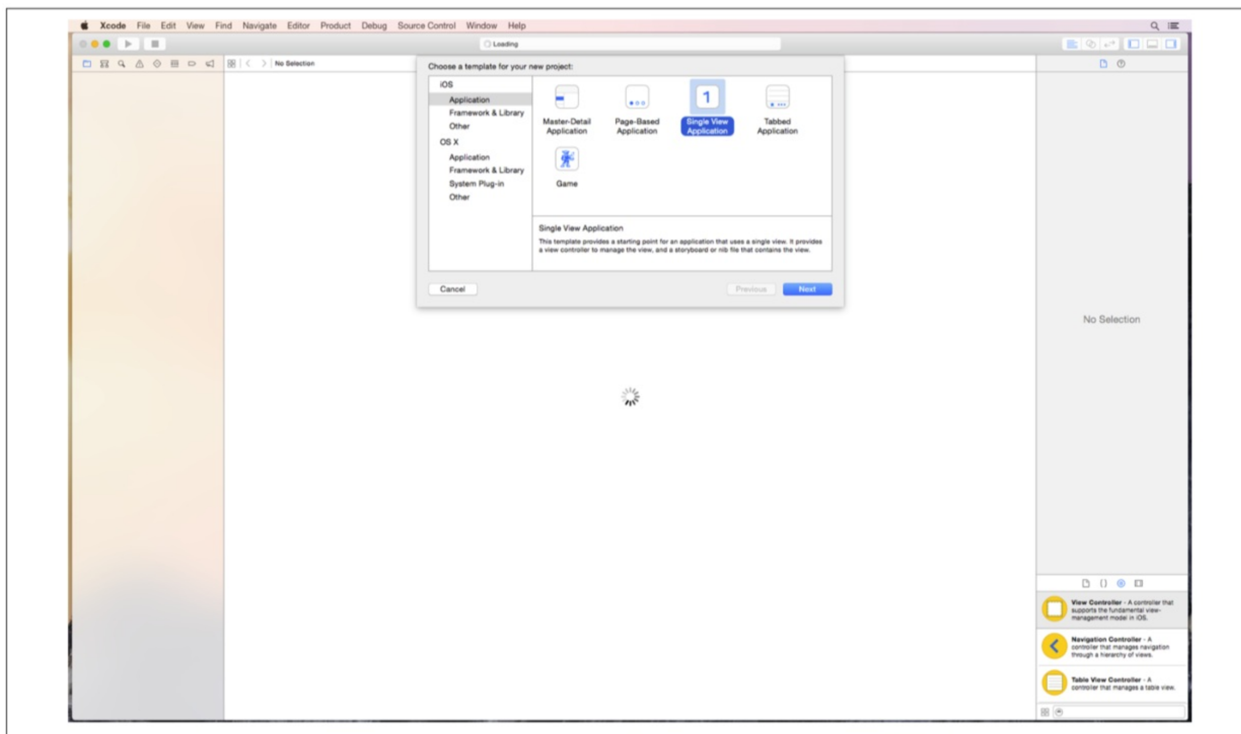
现在你知道了App有哪些功能，那么打开Xcode吧，点击Dock中的Xcode图标，如果Dock中没有Xcode，那么去Spotlight中搜索一下。

接着会出现熟悉的Xcode欢迎界面。关闭欢迎界面，点击顶部菜单栏的File -> New -> Project（见图3-6）。



Page 72 | Chapter 3 : Diving into Swift

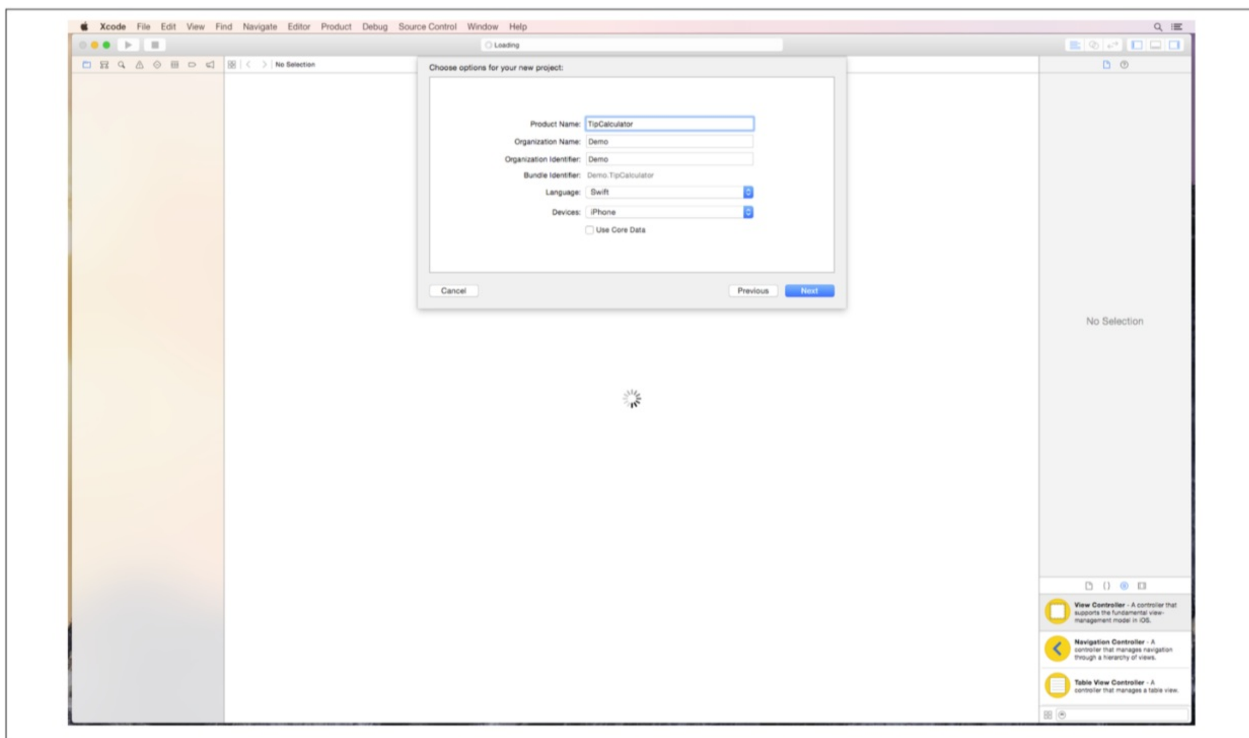
从模板中选择Single View Application，点击Next（见图3-7）。



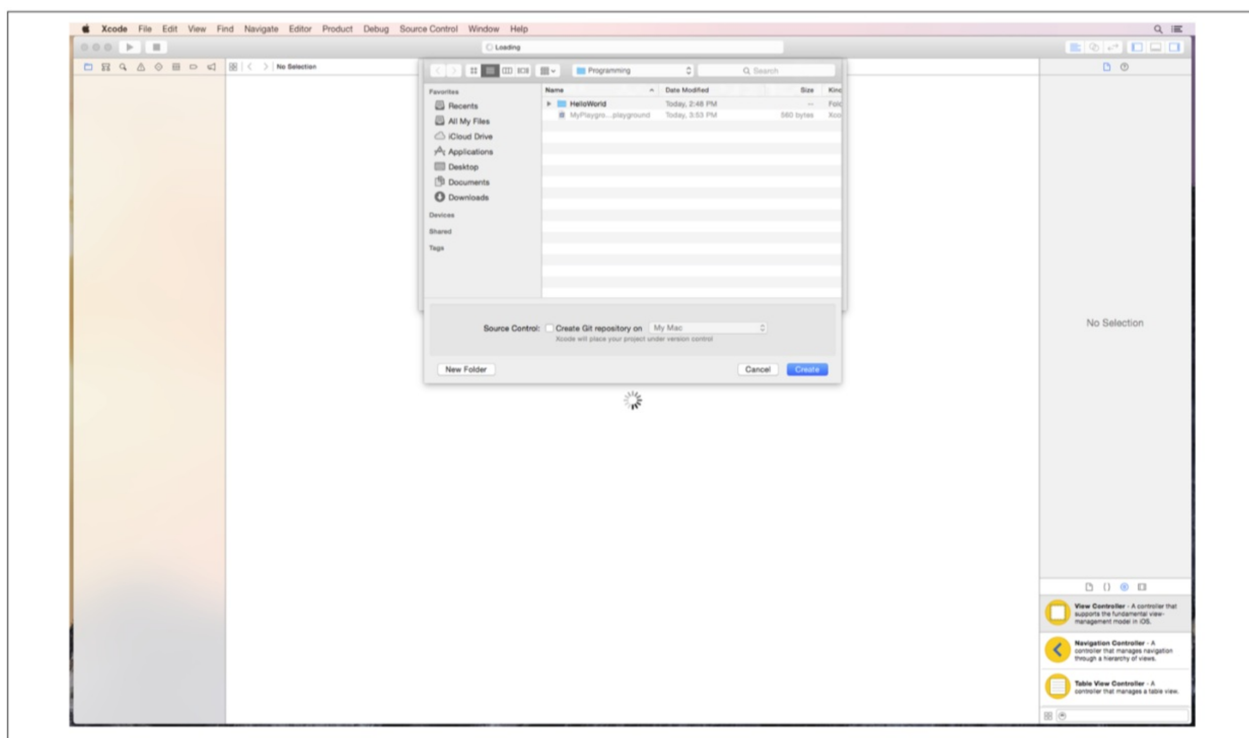
Project的一些项目可能已经自动为你填写好了。

> Exercise: Tip Calculator | Page 73

在Product Name一栏输入TipCalcualtor，Organization Name和Organization Identifier可以写成你的姓名，之间不要有空格。最后，Language选择Swift，Devices选择iPhone，点击Next（见图3-8）。

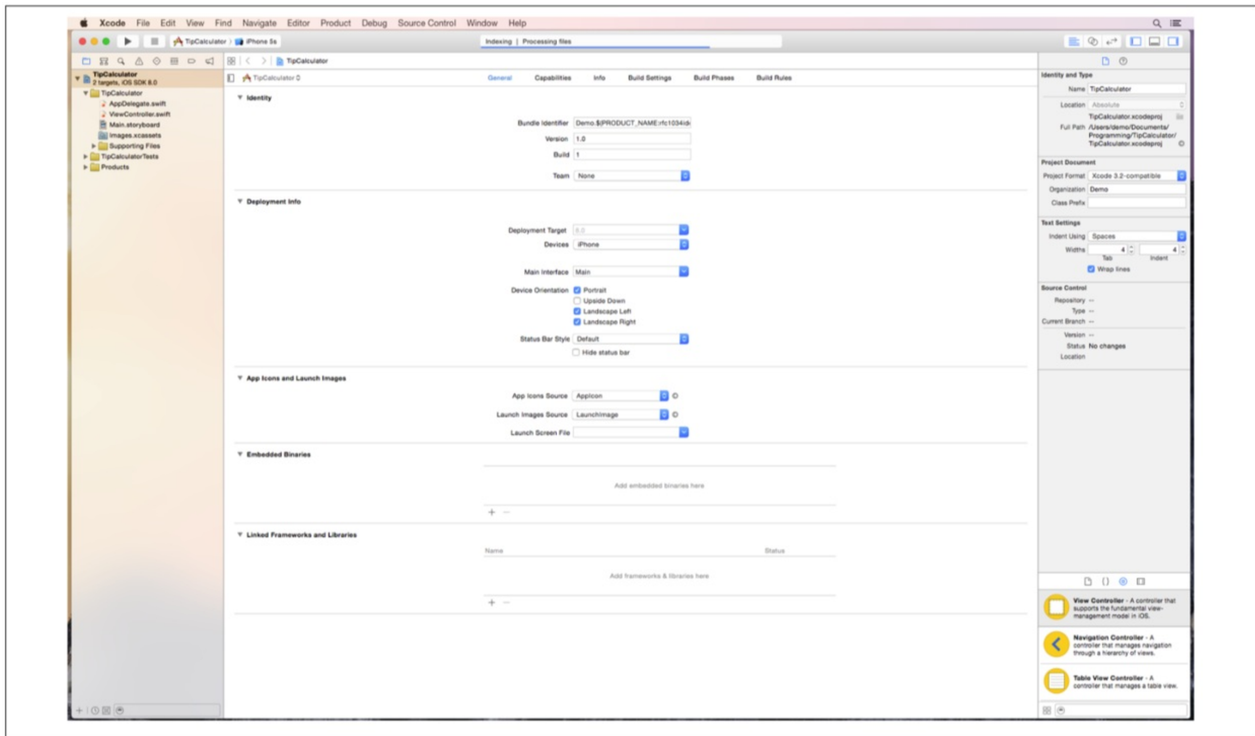


接下来从左侧选择你要存放的文件夹，点击Create，保存工程（见图3-9）。



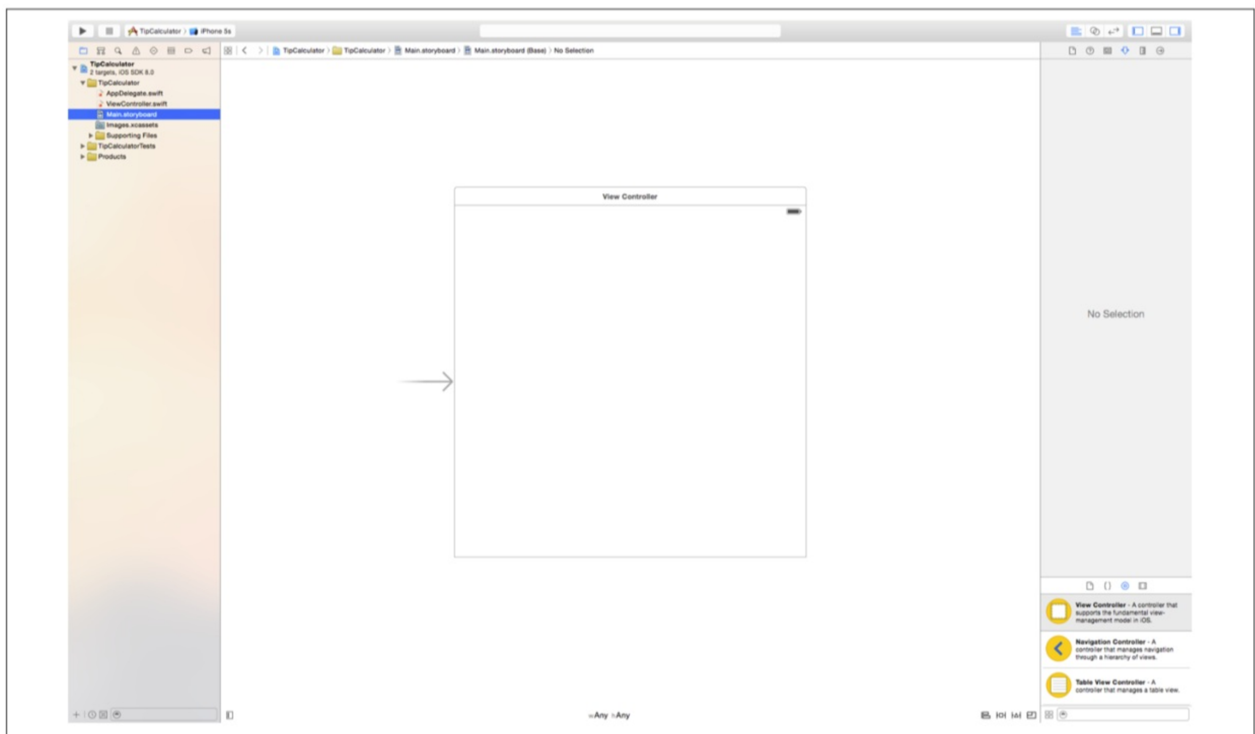
Page 74 | Chapter 3 : Diving into Swift

出现了工程的详细信息界面（见图3-10）。

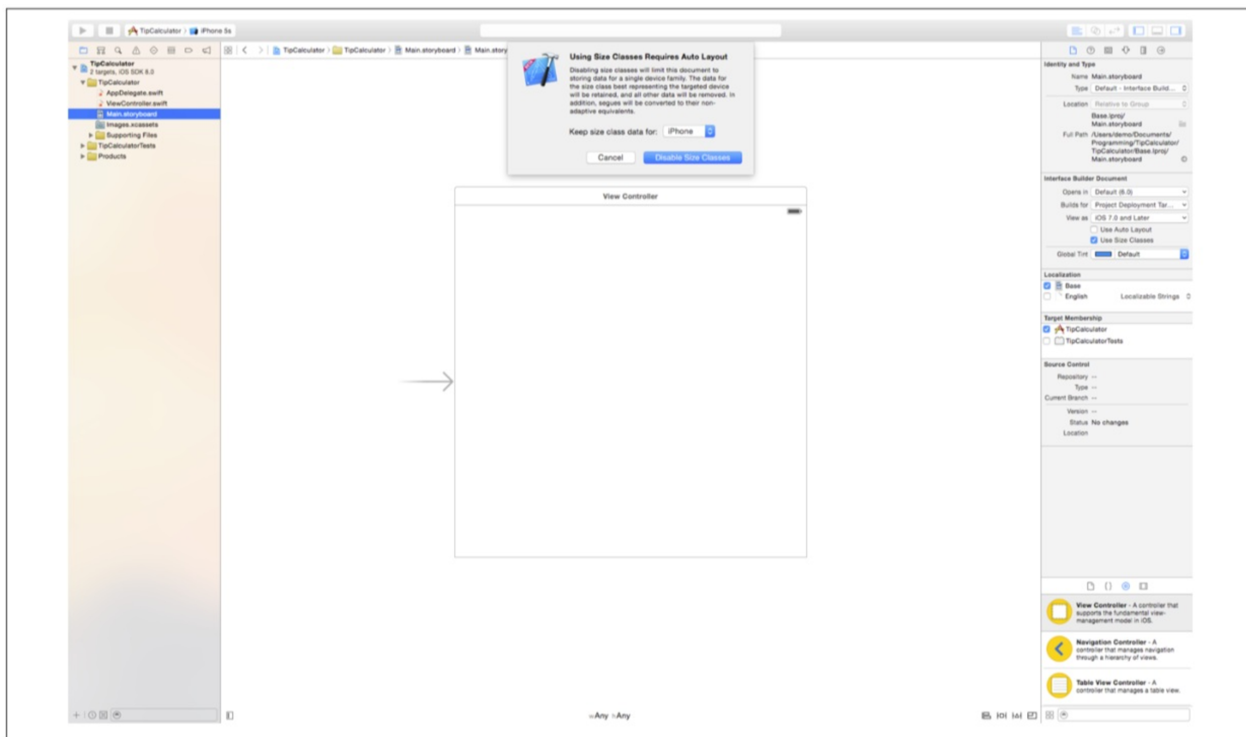


Exercise: Tip Calculator | Page 75

点击Project Navigator中的Main.storyboard文件（见图3-11）。



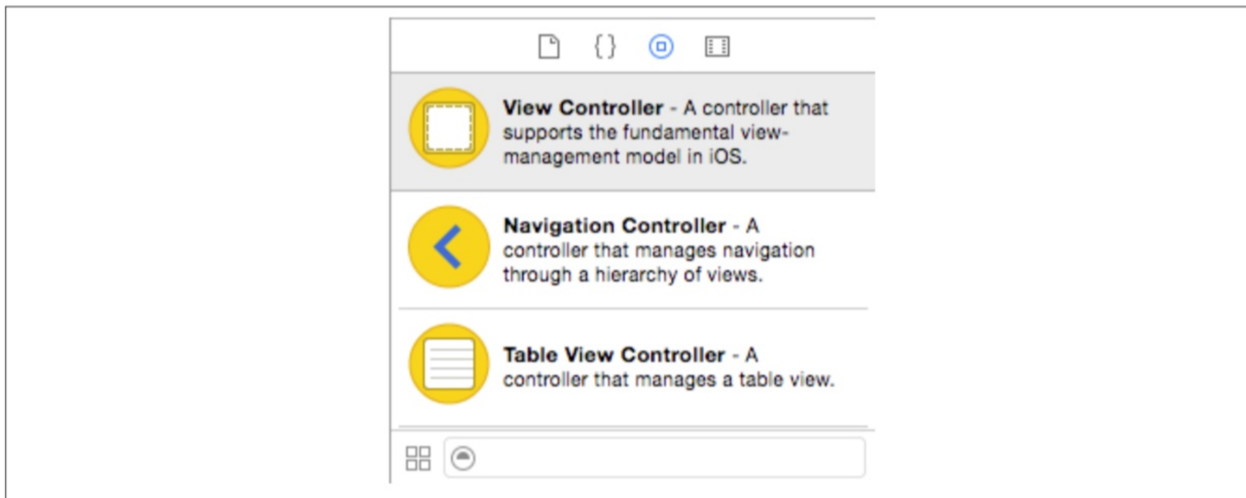
我们这次开发的App仅限于在iPhone上使用，点击Inspector的上方第一个按钮，看起来像是一张纸折了一个角。到中间部分，不勾选Use Auto Layout选项。这时会出现一个对话框，选择iPhone。然后不勾选Disable Size Classes（见图3-12）。这时，Storyboard中的界面形状会改变。我们将会在第七章详细介绍Auto Layout的知识。



Page 76 | Chapter 3 : Diving into Swift

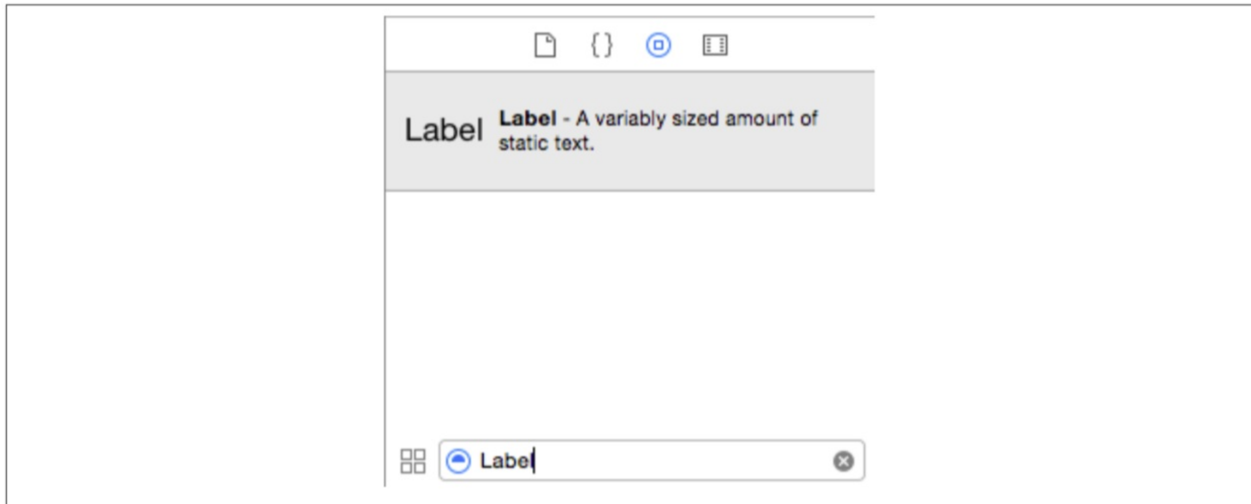
确保Inspector在屏幕的右方。如果没有出现在屏幕右方，点击Inspector View Button，就是右上角右边有一条条纹的按钮，这样就可以显示Inspector了。

Inspector下方有个小的工具栏按钮，Object Library图标（圆圈中有个小方块）在图3-13中蓝色高亮了，如果你的Xcode这个图标不是蓝色，点击一下这个图标。

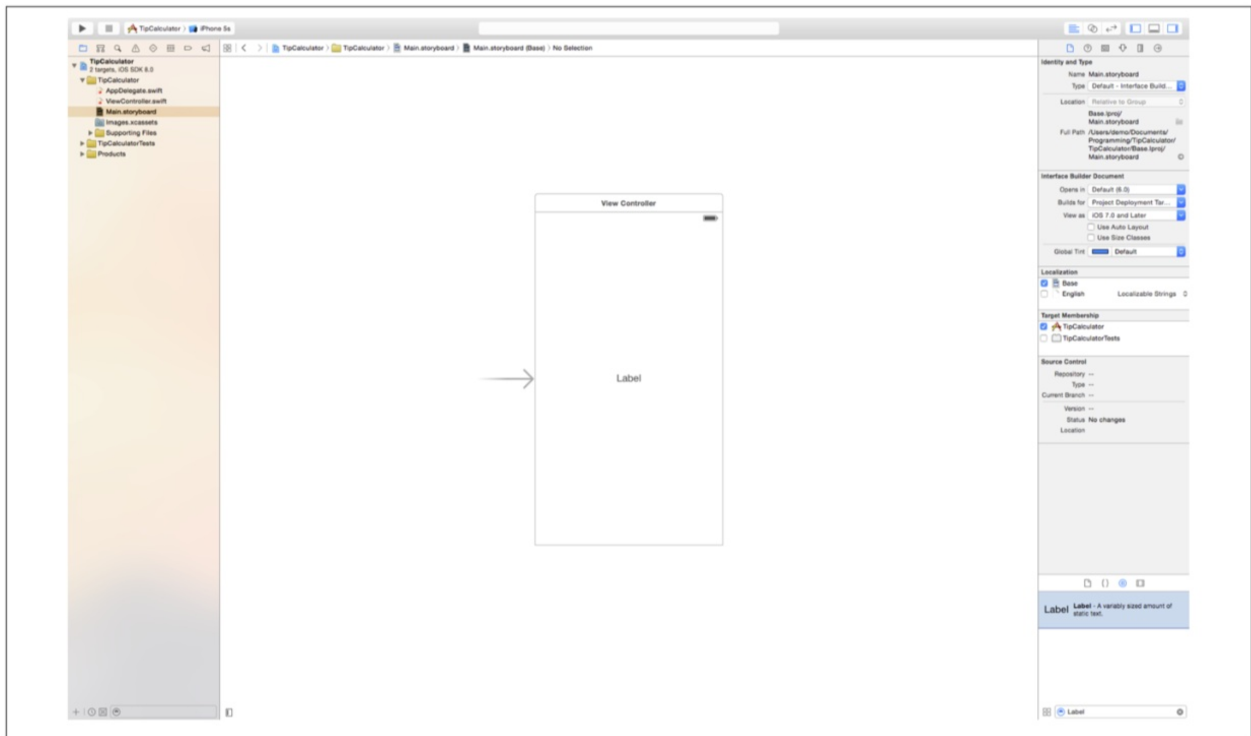


Exercise: Tip Calculator | Page 77

Object Library的下方有个搜索框，输入Label（见图3-14），Object Library搜索框能够方便地找到我们需要的控件。在使用完毕后记得情况输入框中的文字，不然你就看不到所有的控件了。



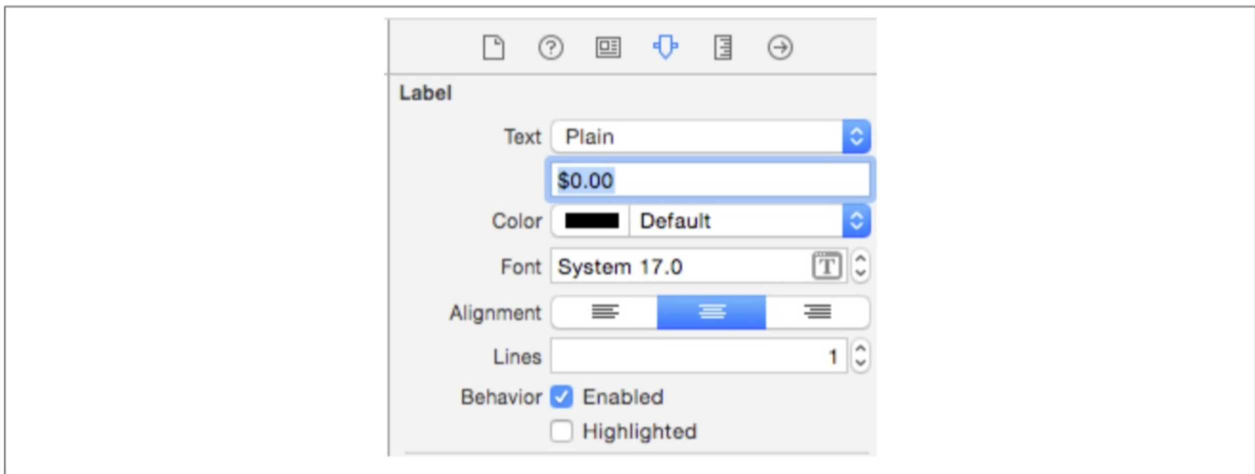
将一个Label控件拖入到界面中（见图3-15）。借助辅助线让Label水平居中且垂直居中。



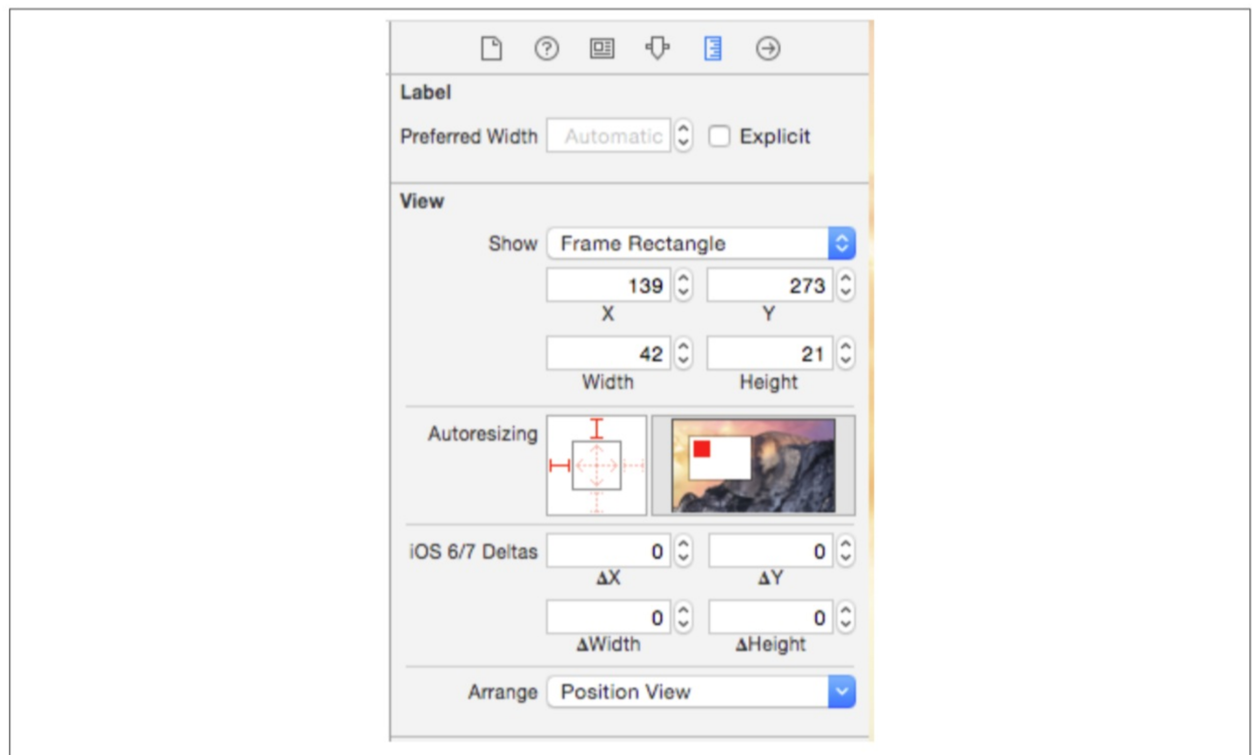
看一下Inspector顶部的工具栏按钮，点击Attribute Inspector图标，从左数第四个（见图3-16），看起来像是一个朝下的箭头。



在text属性中输入为**$0.00**，点击回车。然后点击center alignment按钮（见图3-17）。

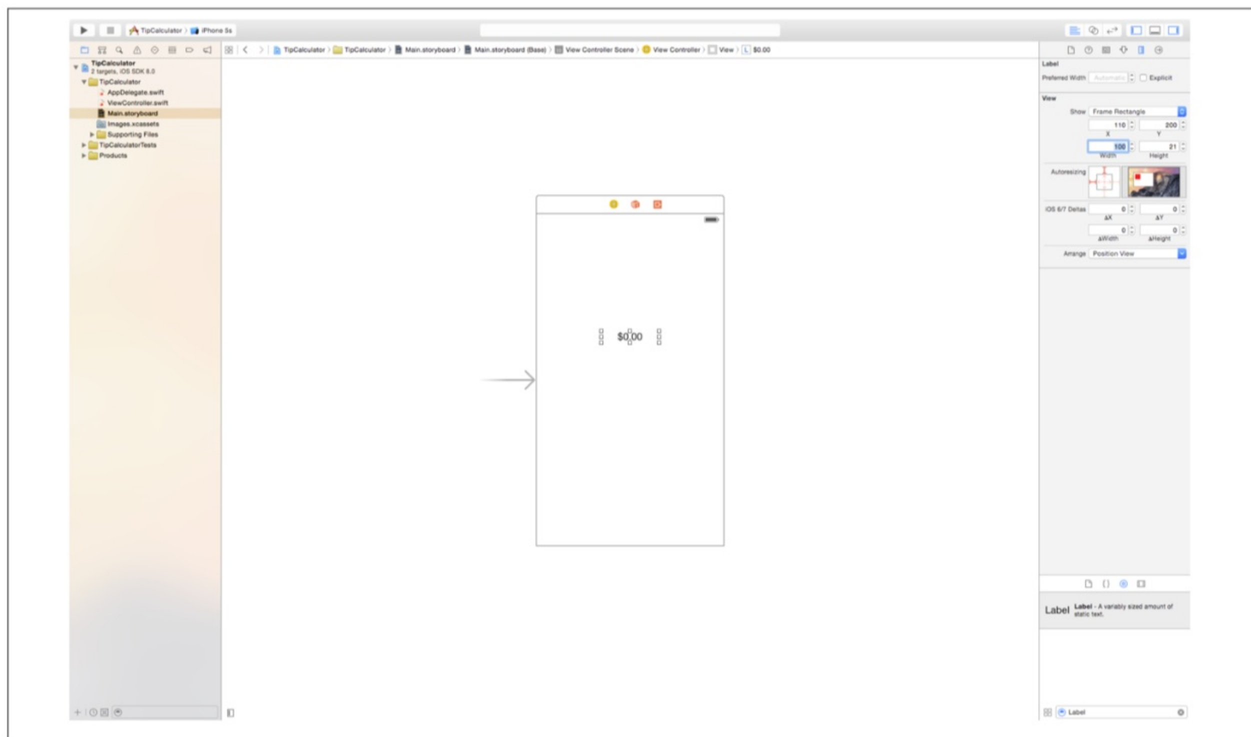


再回到Inspector上方的工具栏，点击从右往左第二个外表像是尺子的图标来打开Size Inspector（见图3-18）。Size Inspector可以设置控件具体位置。



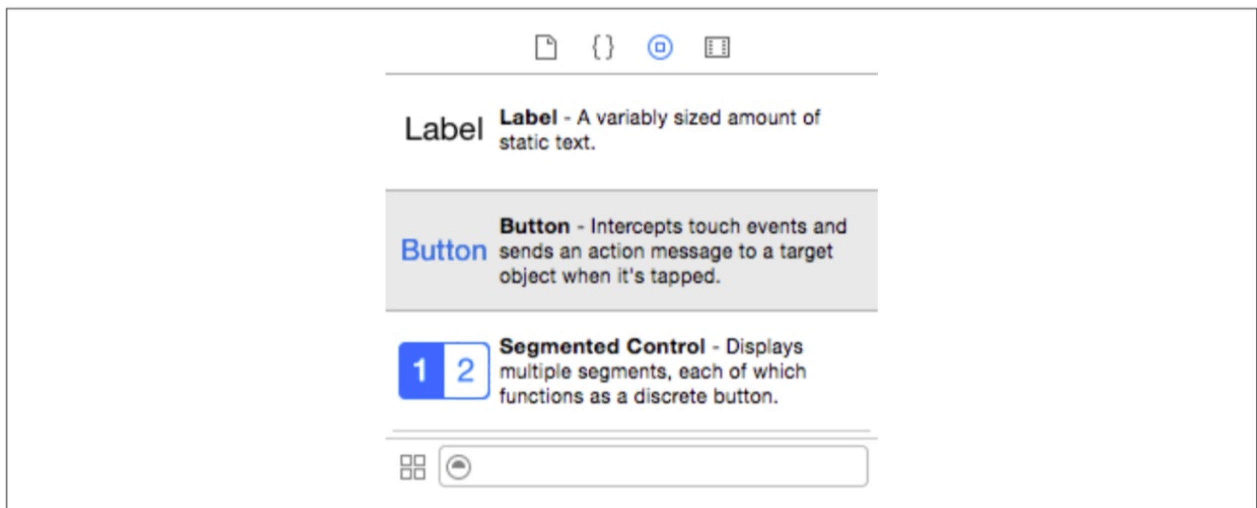
Exercise: Tip Calculator | Page 79

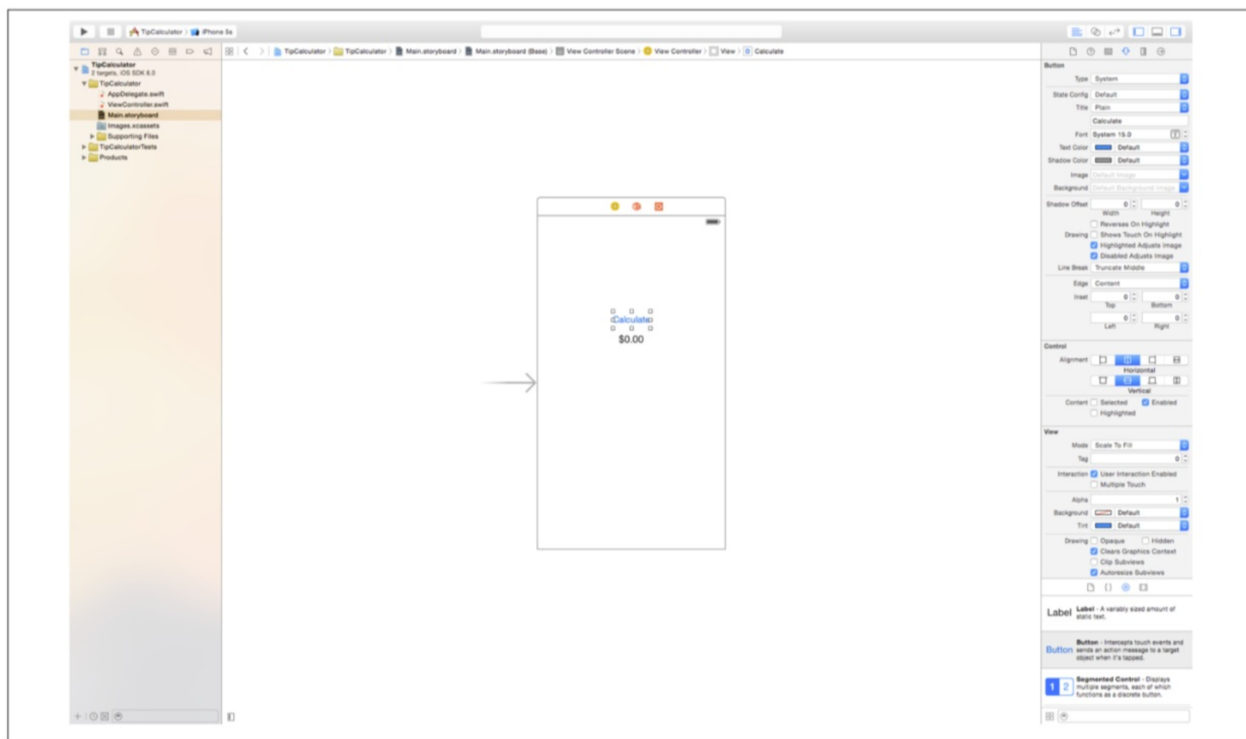
宽设置为100，X值设置为110，Y值设置为200，这时注意界面中Label的位置已经发生了变化（见图3-19）。点击Attribute Inspector图标。



Page 80 | Chapter 3 : Diving into Swift

接下来，我们要往界面上放一个Button。把鼠标光标移动到右下角，在Object Library找到Button控件（见图3-20），把Button控件拖动到界面上。借助辅助线让Button水平居中，放在Label的上方（见图3-21）。

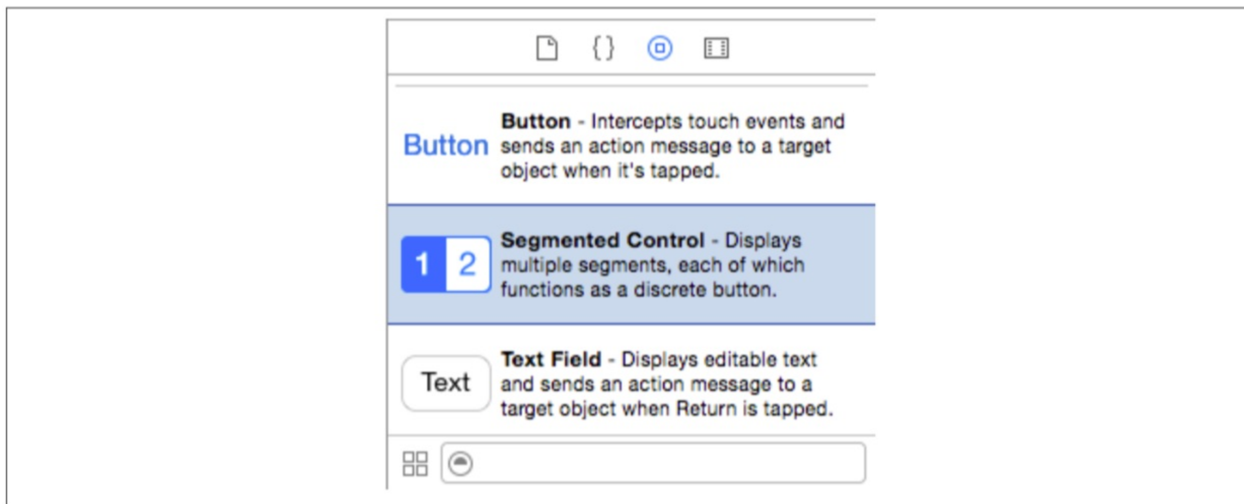




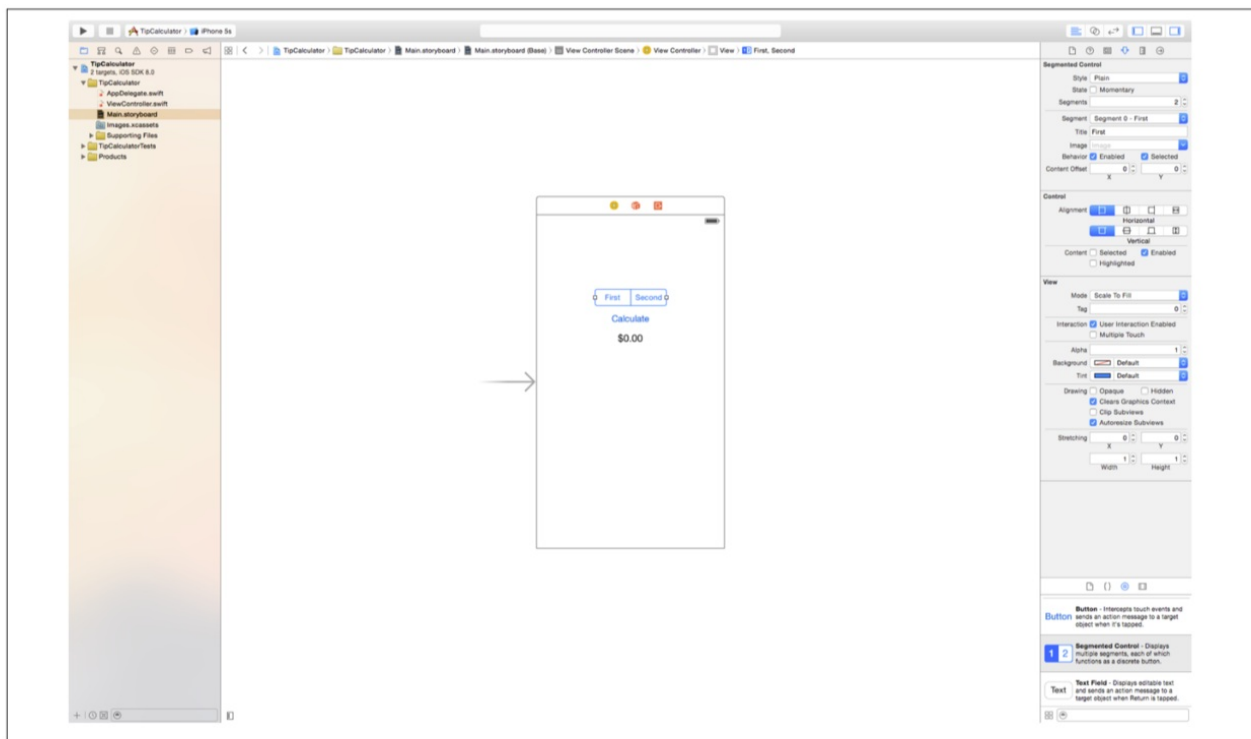
双击Button，输入Calculate，回车。

> Exercise: Tip Calculator | Page 81

接下来我们要添加一个Segmented Control（见图3-22）。Segmented Control有点像是switch不过有更多选择。Segmented Control可以在两个或者更多的segments之前点击切换，一次只能选择一个segment。

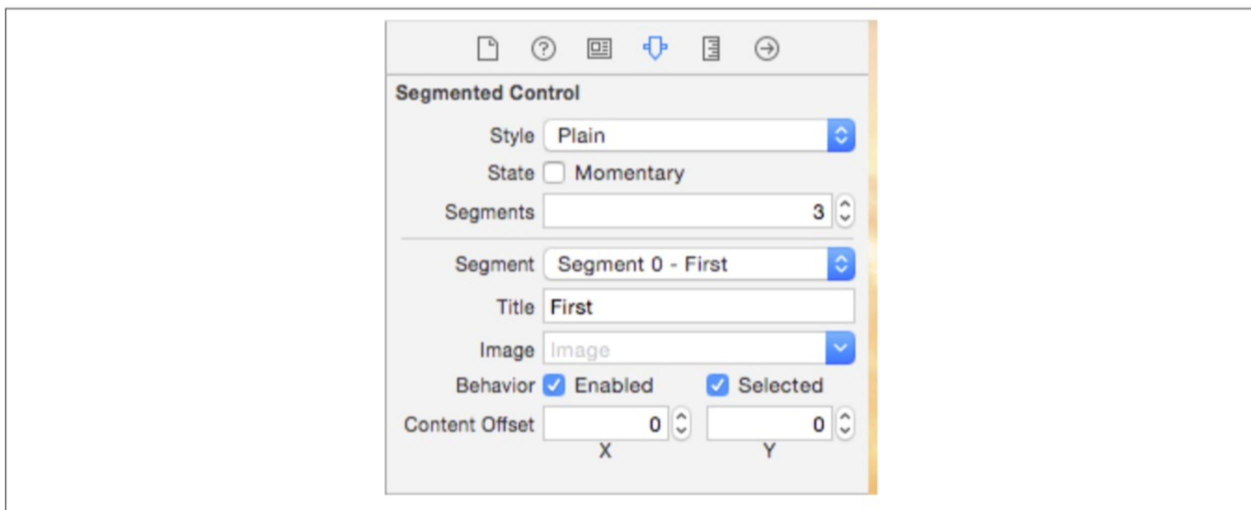


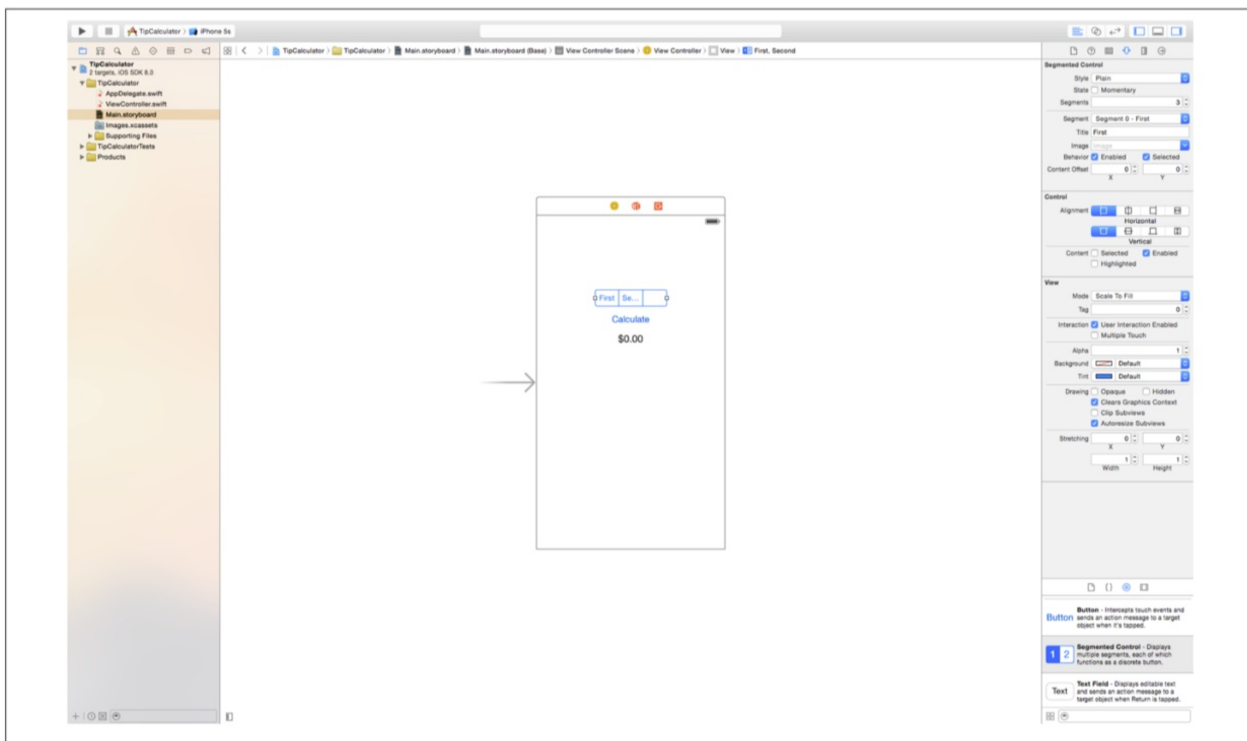
在Object Library中找到Segmented Control，然后拖到界面上，把Segmented Control放到Button的上方（见图3-23）。



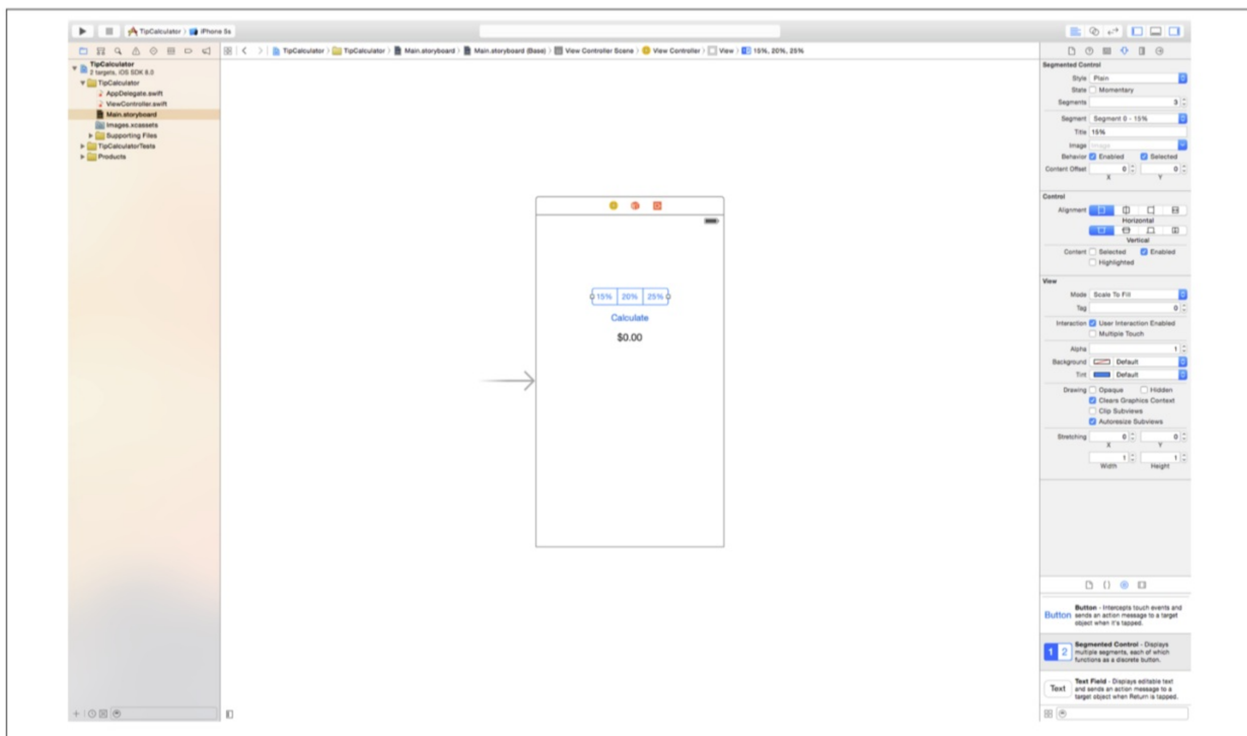
Page 82 | Chapter 3 : Diving into Swift

选中Segmented Control，然后点击Inspector上的Attribute Inspector，把Segments的数字修改为3（见图3-24）。你会看到界面上的Segmented Control已经变成了3个（见图3-25）。





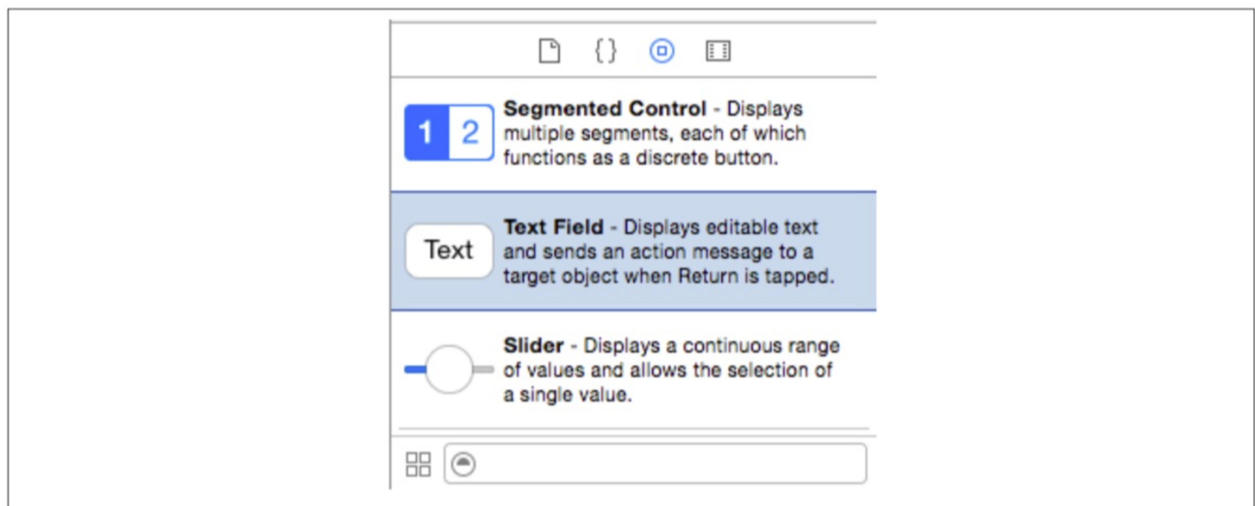
接着双击Segmented Control上的First单词，修改为15%，回车。双击Second修改为20%，双击第三个Segment，输入25%（见图3-26）。

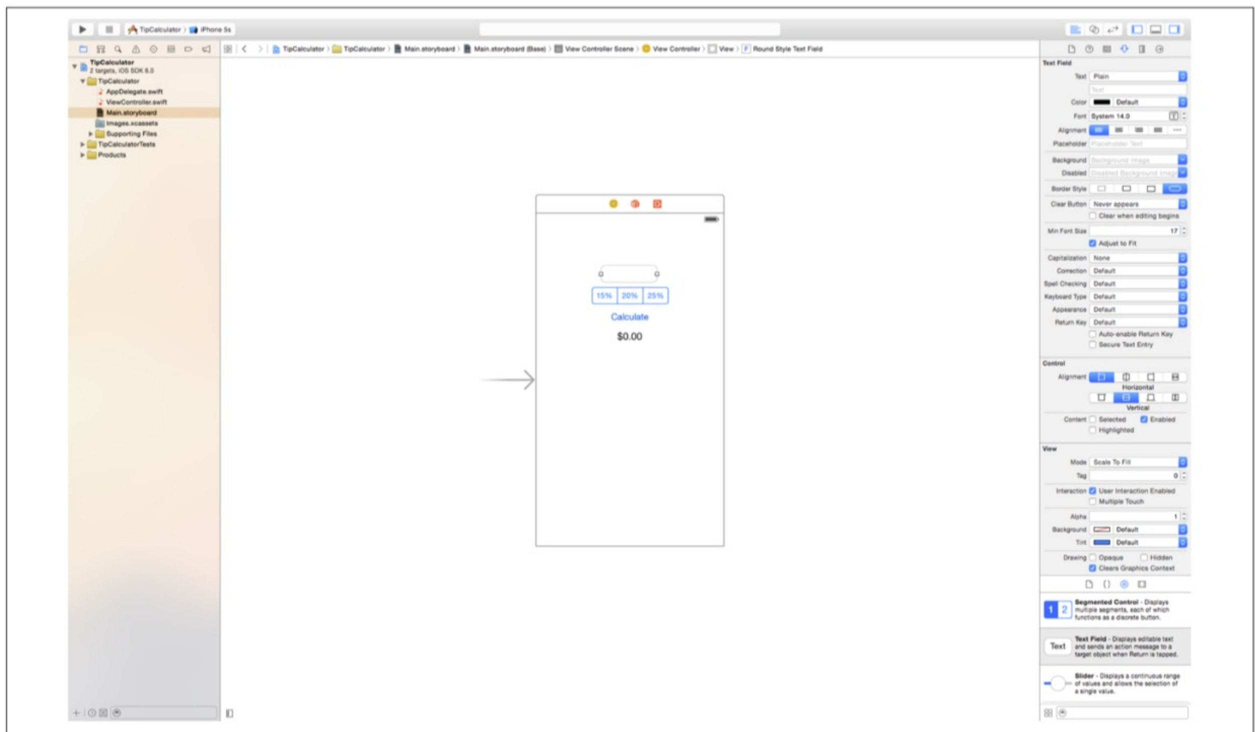


> Exercise: Tip Calculator | Page 83

最后需要添加的控件是Text Field，Text Field可以让用户通过敲击键盘输入东西。

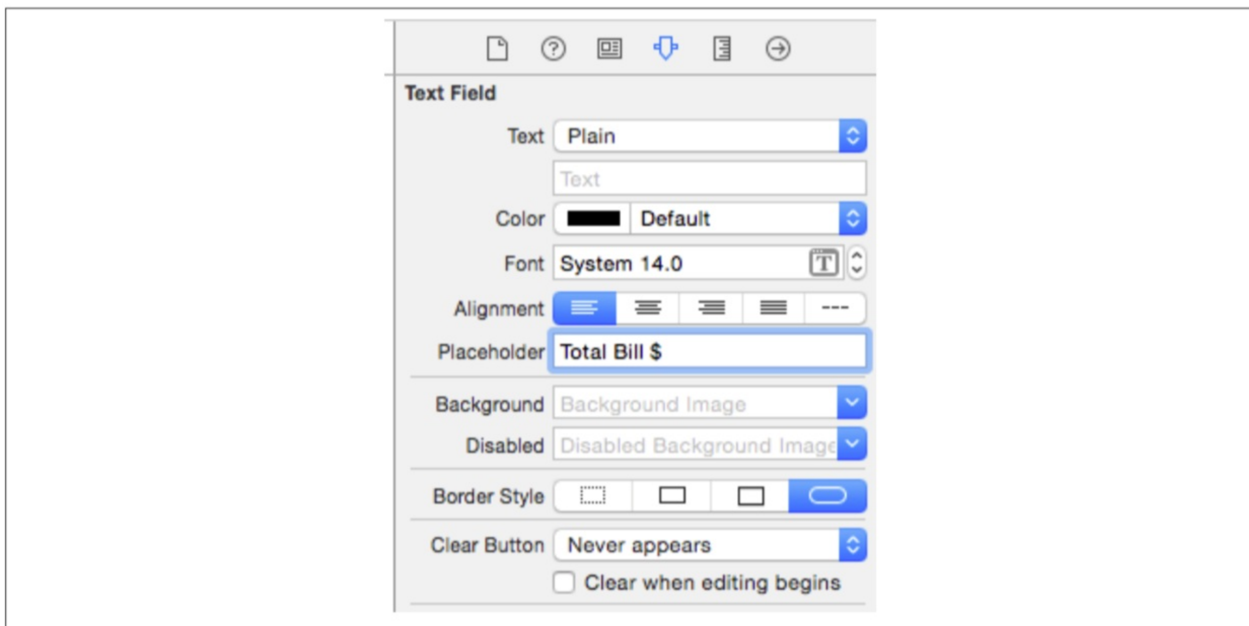
到Object Library中找到Text Field（见图3-27）。把Text Field拖到界面上，放到Segmented Control的上方（见图3-28）。





Page 84 | Chapter 3 : Diving into Swift

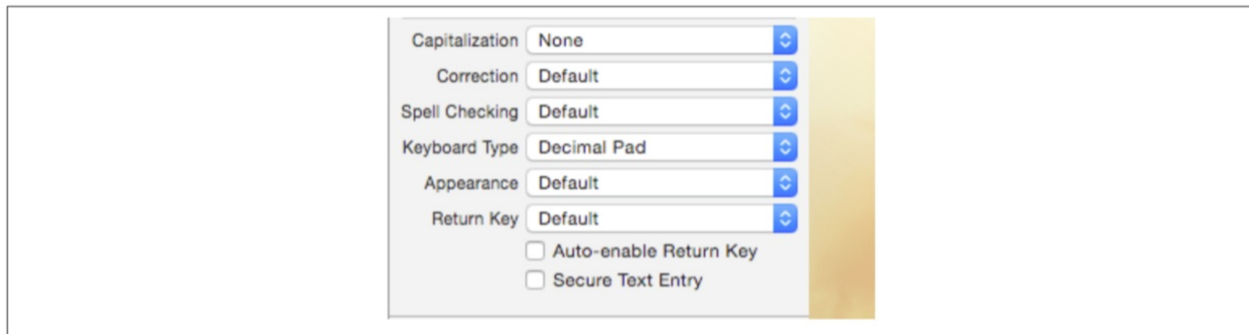
点击Attribute Inspector，找到Placeholder属性，从上往下第五行。输入Total Bill $（见图3-29）。placeholder的文案有点像是向导，指导你在Text Field中输入什么文字。



当用户点击Text Field时，placeholder中的文字会消失不见，然后键盘出现在屏幕上。

> Exercise: Tip Calculator | Page 85

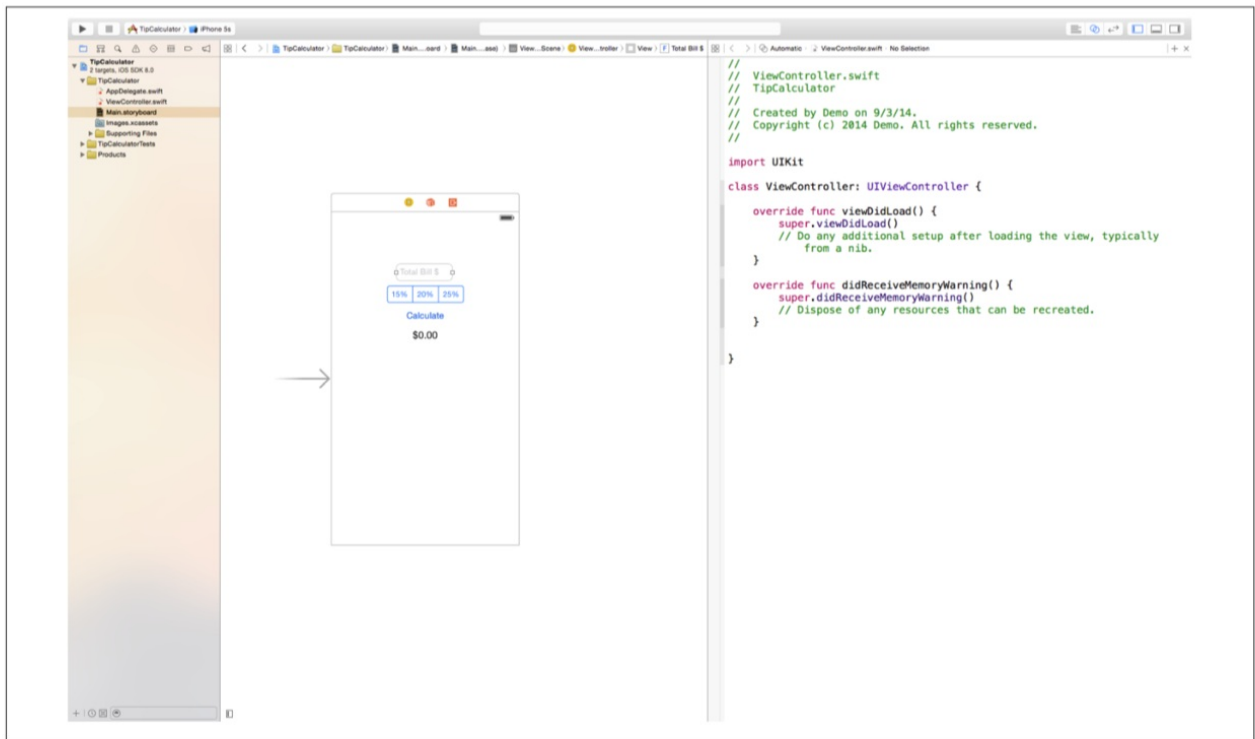
接下来在Inspector中找到Keyboard Type属性，点击Default右边下拉按钮，选择Decimal Pad（见图3-30）。这样，弹出的键盘就只有数字了，用户不需要在这里输入字母。



现在界面已经搭建完毕了，那么接下来需要我们把界面view和conroller（就是swift文件）连起来了。和上一章节中的练习一样，我们会用到Assistant Editor，来连接view和controller。

在屏幕的右上角，点击Assistant Editor按钮（见图3-31）。打开Assistant Editor后，点击Inspector View按钮，让Editor的空间更大一些（见图3-32）。





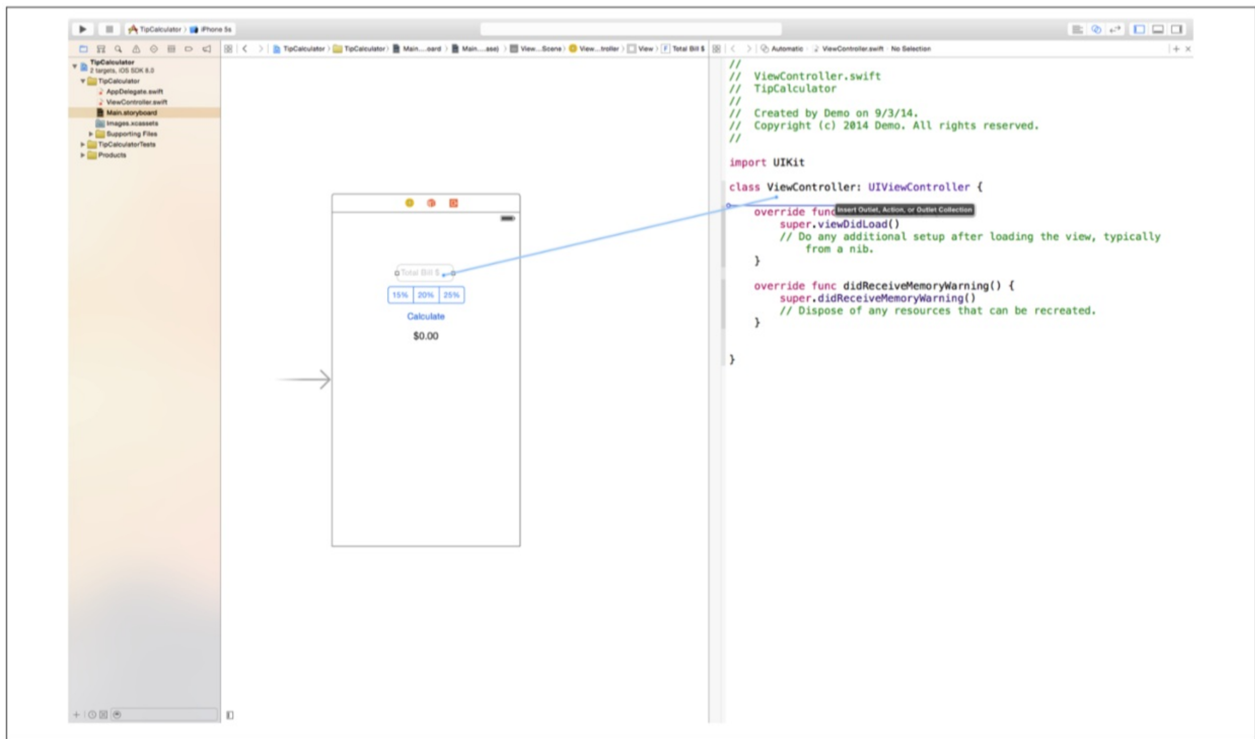
Page 86 | Chapter 3 : Diving into Swift

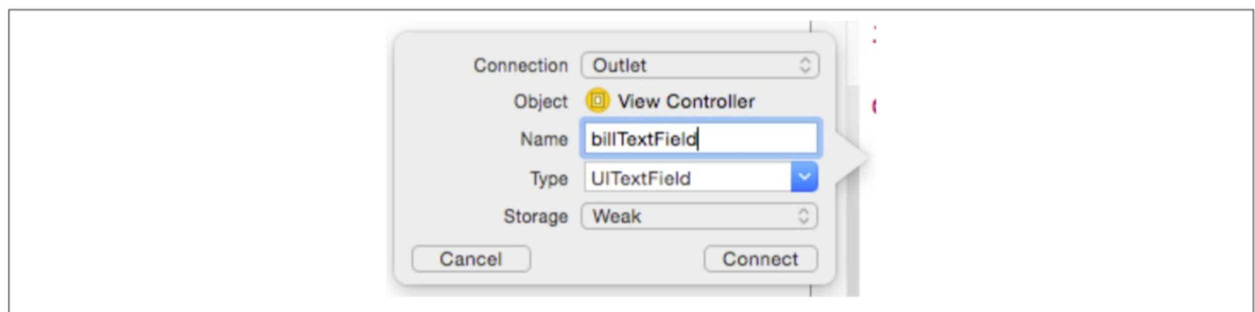
Assistant Editor会自动打开ViewController.swift文件，Assistant Editor上方在“Automatic”右边会有文件名称，核对一下是否是ViewController.swift。

选中界面上的Text Field控件，同时按住Control键，将Text Field控件拖到右边ViewController.swift文件中下面这行代码的下方：

```
class ViewController: UIViewController {
```

当你看到一条蓝色的横线时，然后松开鼠标（见图3-33），接着出现一个弹出框（见图3-34）。



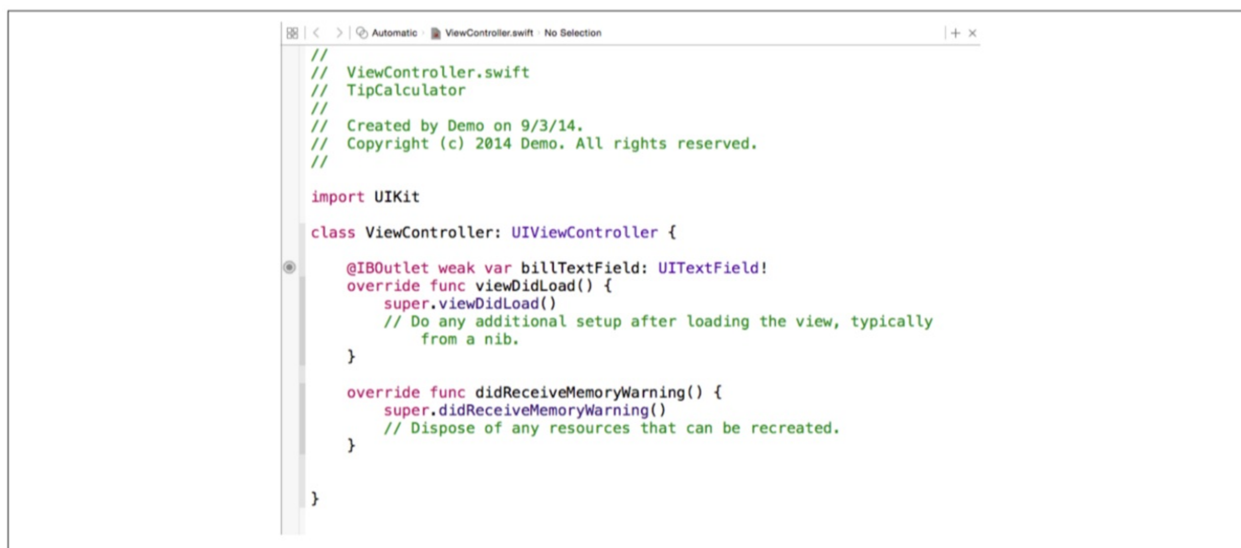


Exercise: Tip Calculator | Page 87

因为你要获取Text Field中用户输入的数值，所以我们使用Outlet连接类型。在Name一栏中输入billTextField作为变量名，其余不变，点击Connect完成连接。

当你点击Connect后会产生一行代码（见图3-35）。看一下这行代码：

```
@IBOutlet var billTextField : UITextField!
```



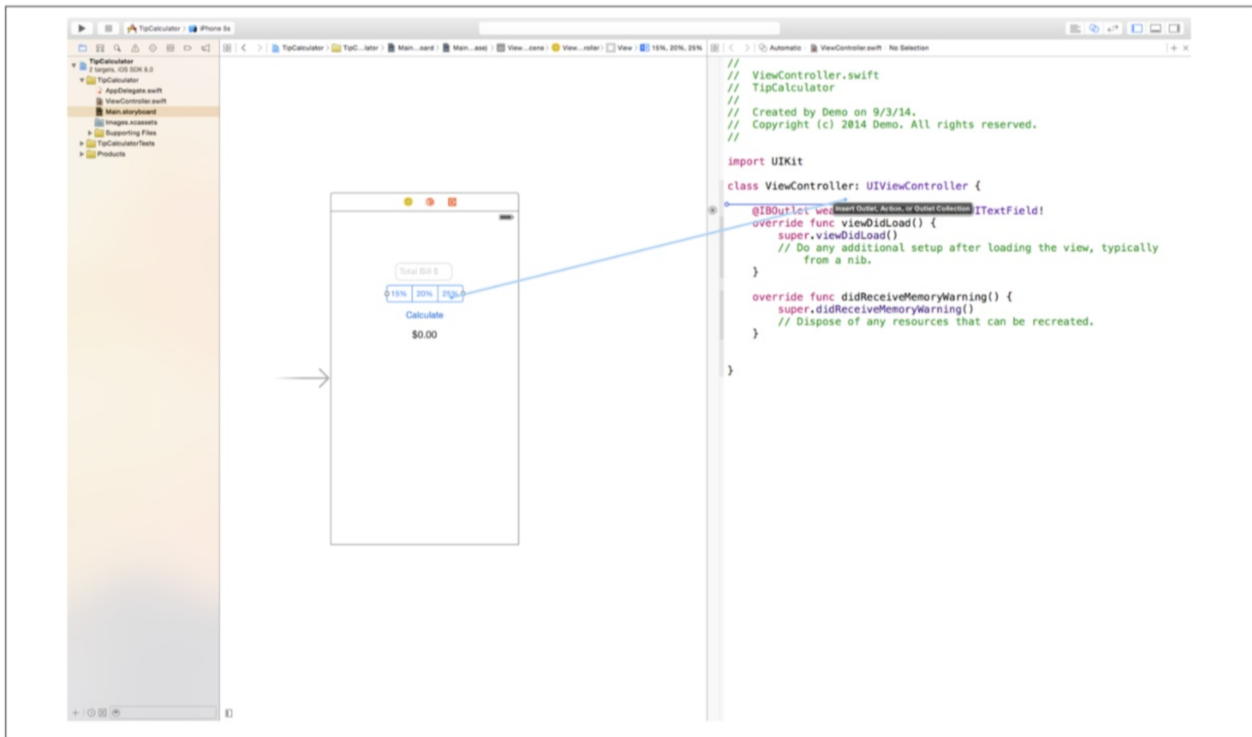
Page 88 | Chapter 3 : Diving into Swift

我们先跳过第一个单词，过会再来讨论它。先看一下var这个单词，var用来声明变量，billTextField是这个变量的名字，冒号用来声明这个变量的类型，UITextField就是这个变量的类型。最后，@IBOutlet这个次用来表示这是view和controller之间的一个outlet连接。billTextField这个变量在代码就代表界面上的这个Text Field控件。

选中Segmented Control控件，同时按住Control键，然后拖到右边ViewController.swift文件中下面这行代码的下方：

```
class ViewController: UIViewController {
```

当你看到一条蓝色的横线时，然后松开鼠标（见图3-36），接着出现一个弹出框（见图3-37）。





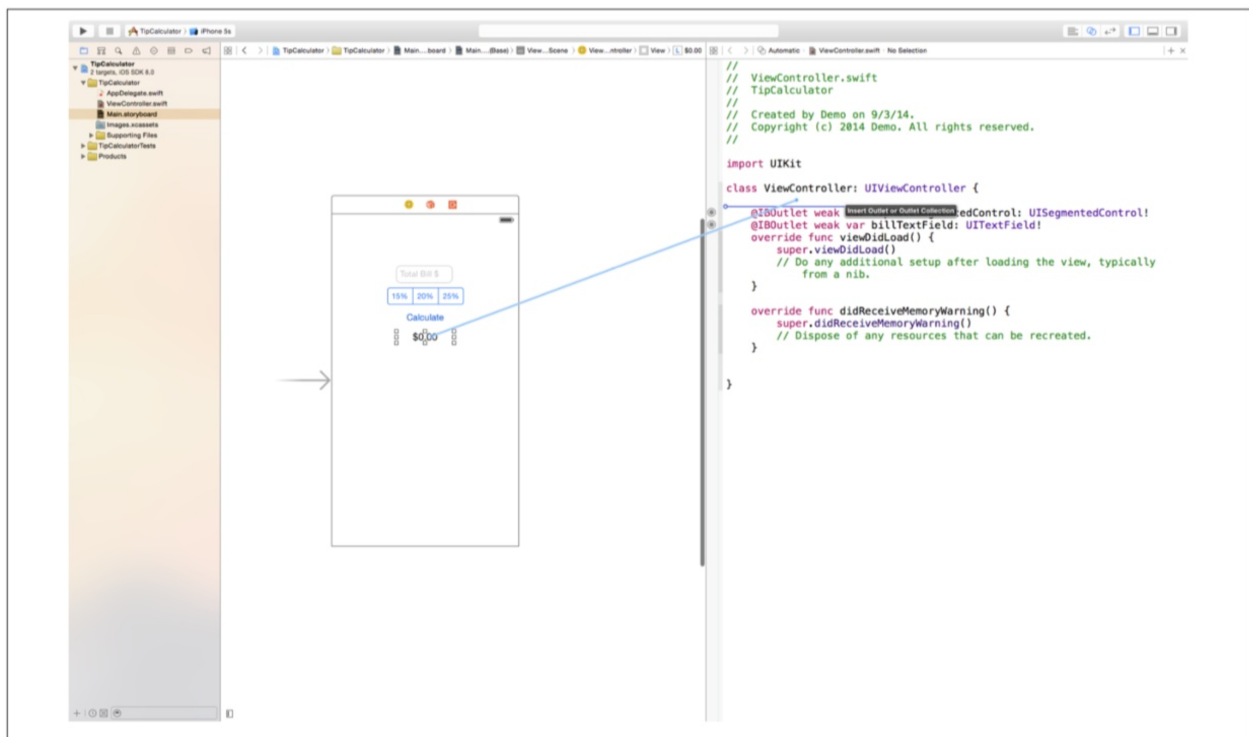
Exercise: Tip Calculator | Page 89

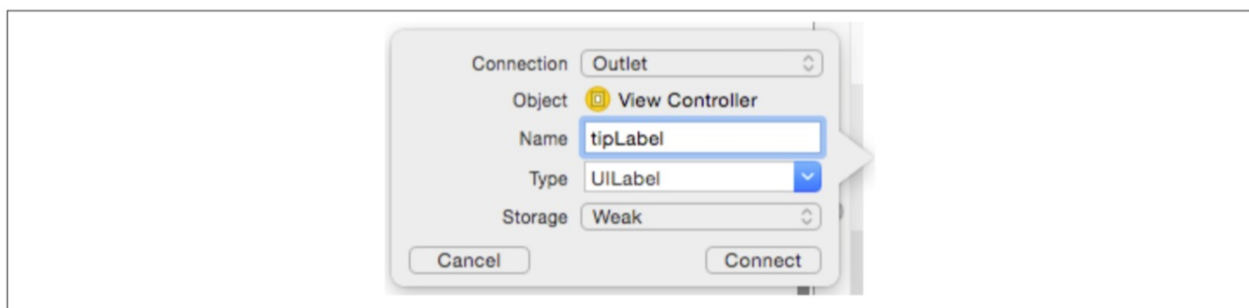
将Connection设置为Outlet，Name是tipRateSegmentedControl。剩下的选项不变，然后点击Connect。

接下来选中界面上的Label控件，同时按住Control键，然后拖到右边ViewController.swift文件中下面这行代码的下方：

```
class ViewController: UIViewController {
```

当你看到一条蓝色的横线时，然后松开鼠标（见图3-38），接着出现一个弹出框（见图3-39）。





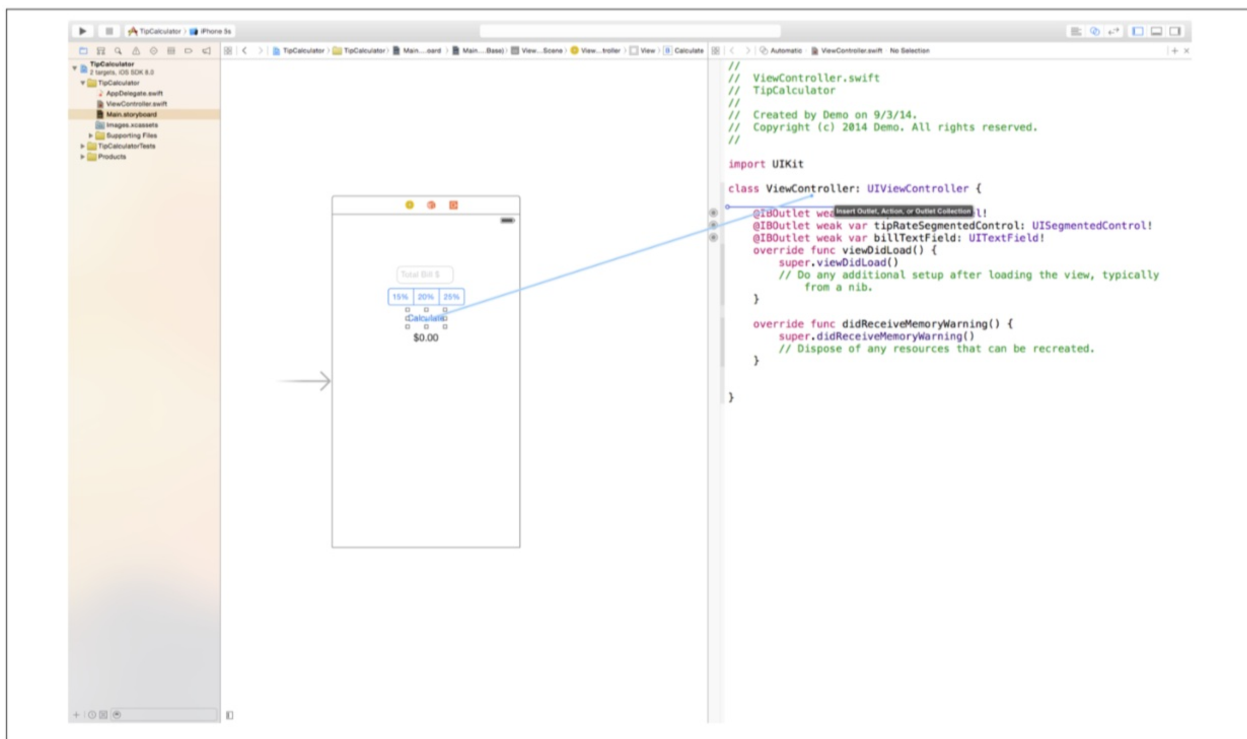
Page 90 | Chapter 3 : Diving into Swift

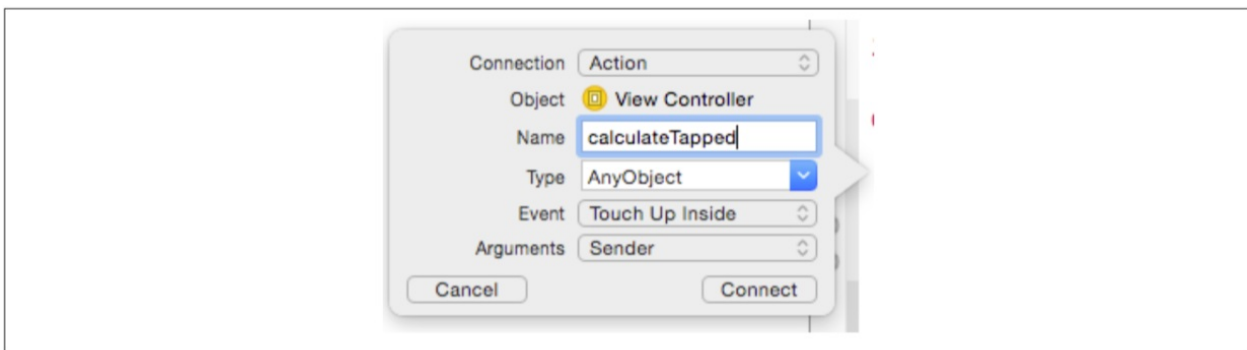
将Connection设置为Outlet，Name是tipLabel，剩下的选项不变，然后点击Connect。

最后，选中界面上的Button控件，同时按住Control键，然后拖到右边ViewController.swift文件中下面这行代码的下方：

```
class ViewController: UIViewController {
```

当你看到一条蓝色的横线时，然后松开鼠标（见图3-40），接着出现一个弹出框（见图3-41）。





> Exercise: Tip Calculator | Page 91

将Connection设置为Action，Name是calculateTapped。剩下的选项不变，然后点击Connect。

你添加到ViewController.swift中的代码会是下面这个样子：

```
@IBOutlet var tipLabel : UILabel!
@IBOutlet var tipRateSegmentedControl : UISegmentedControl!
@IBOutlet var billTextField : UITextField!

@IBAction func calculateTapped(sender : AnyObject) {

}
```

> Page 92 | Chapter 3 : Diving into Swift

calculateTapped action会自动添加一个方法，每次用户点击Button时，这个方法都会被调用。把你的鼠标光标放到这个方法的两个大括号之间。

按照最佳实践原则，我们需要写注释，说明你在接下来的代码中将要完成哪些事情。这个过程，就做pseudocoding（伪代码），能够让你在写一个方法之前有一个大概的轮廓。你可以直接在Xcode中写注释，注释还可以帮助其他程序员看懂你的代码。当运行App或者编译代码时，我们是看不到注释的。

前面先写两个斜杠，后面的文字就是注释了。例如：

```
@IBAction func calculateTapped(sender : AnyObject) {
    //This code is run each time the Calculate Button is tapped.

}
```

在calculateTapped方法中添加下面的注释：

```
@IBAction func calculateTapped(sender : AnyObject) {
    //1\. 获取账单的总金额
    //2\. 确定小费费率
    //3\. 计算小费金额
    //4\. 将消费金额呈现给用户

}
```

接下来，需要确定费率：15%，20%或25%。确定下费率后，总金额乘以费率，就是小费金额，然后将消费金额呈现给用户。

现在你已经给方法大体规划了计划，那么我们需要为每一步编写代码了。

想要获得总金额，使用要使用Text Field控件变量billTextField的text属性。把鼠标光标放到第一行注释的后面，按回车。接着输入下面的代码：

```
Introducing IOS 8
var userInput = billTextField.text as NSString
```

（注意，作者写这本书的时候swift2.0还没有出来，现在NSString和String合二为一了，苹果官方推荐使用String，不再推荐使用NSString了）

你可以注意到了，在你输入代码的时候，Xcode会提供一些单词来帮助你写代码，例如，从billTextField中删除“.text”，然后只输入“.”。

这种预测输入系统叫做Autocomplete（自动补全）。自动补全极大的提高了开发者的体验，自动补全弹出框会显示此变量所有可用的属性和方法。

> Exercise: Tip Calculator | Page 93

在billTextField.后面在输入te两个字母，自动补全就会更新可用属性，甚至会显示text属性返回一个字符串，点击Tab键接受自动补全。

现在你已经获取了Text Field控件中的text，然而，text是一个字符串格式，不是数值，无法进行数学计算，我们需要把text的字符串类型转换为Float类型。Float是带有小数点的数字，Float类型特别适合处理货币计算。转换类型的代码如下：

```
var totalBill: Float = userInput.floatValue
```

这行代码创建了一个新的变量totalBill，然后设置类型为float。同时，把用户输入的字符串转换为浮点型然后赋值给了totalBill变量，使用的 `floatValue` 方法。第一步完成了。

接下来是确定小费费率，要检测用户选择的segments三个选项中哪一个，这样就确定了费率。Segmented Control有个属性是selectedSegmentIndex，这个属性能够告诉你用户现在选中的是哪个segment。在第二个注释后面按回车然后添加下列代码：

```
var index: Int = tipRateSegmentedControl.selectedSegmentIndex
```

这行代码创建了一个叫做index的新变量，同时把tipRateSegmentedControl选中的那个segment的值赋值给了index变量。index以零开头，index 0就是15%，index 1就是20%，index 2就是25%.

现在你已经知道了选中了哪个segment，你需要根据选中的segment得到对应的费率，这个地方适合用if条件句，在你写if条件句之前，先创建一个变量存储当前的费率：

```
var tipRate: Float = 0.15
```

这行代码创建一个浮点型变量tipRate。tipRate的默认值为0.15，tipRate会根据不同的segment转换不同的费率。在代码中写下下面的if语句：

```
var tipRate: Float = 0.15
if  index==0  {
     tipRate = 0.15
}
```

> Page 94 | Chapter 3 : Diving into Swift

if条件句用if开头，接下来跟着条件。在这里，条件就是检查index的值是否是0，如果是0，那么返回true，如果不是0，返回false。如果条件为true，那么两个大括号直接的代码就会被执行，否则会跳过大括号里的代码，执行下一个条件。给if语句增加else if：

```
var tipRate: Float = 0.15

if index==0 {
    tipRate = 0.15
} else if index==1 {
    tipRate = 0.20
}
```

else if检查另外的条件，如果index是0，那么else if语句就不会执行，else if检查index的值是不是1，如果是1则为true，就会把0.20赋值给tipRate。增加if语句最后一部分：

```
var tipRate: Float = 0.15

if index==0 {
    tipRate = 0.15
} else if index==1 {
    tipRate = 0.20
} else {
    tipRate = 0.25
}
```

最后这部分，你添加了else语句，如果前两个if条件都不符合，那么就会触发else语句。在这里，index的值只有三种可能：0，1，2。else处理的是如果index为2的条件。

现在你已经有了总金额和小费费率，可以进行计算了。把你的鼠标光标放到第三行注释后面按回车，写下下列代码：

```
var tip: Float = totalBill * tipRate
```

这行代码创建了名为tip的浮点型变量，totalBill乘以tipRate后的值赋值给tip。这样，我们就完成了第三步。

把鼠标光标放到第四行注释后面按回车，然后输入下方的代码：

```
tipLabel.text = "$\\(tip)"
```

这行代码设置了tipLabel的text属性，我们用双引号来创建新的字符串，在字符串里面，先放一个美元的货币符合，然后()这部分是起着占位符的作用，可以存放任何变量，在这里，占位符中是tip变量。

这时点击Xcode的Play按钮（或者Command+R），iOS模拟器就会启动然后运行程序，App启动后，点击Text Field，输入一个数字，然后选择一个费率，点击Calculate按钮，Label中就会出现你需要付出的小费金额。

> Exercise: Tip Calculator | Page 95

如果App没有按照你想要的结果运行，或者程序有了错误或警告，不要太担心，学习的最佳方式就是试错，熟能生巧，到我们的网站上下载示例代码，对比一下代码，多试几次，直到搞定这个程序为止。

> Page 96 | Chapter 3 : Diving into Swift

第四章：深入了解Swift

在这一章节中，我们会更深入的了解Swift，苹果公司给开发者提供了一系列的方法和类，也就是常说的framework（框架），这些框架叫做Foundation。你可以从developer.apple.com上找到更多关于苹果公司框架的知识。然而，这些框架也有不够用的时候，这就需要你能够编写你自己的类（class）或方法（methods）。在这一章节中，我就学习如何编写你的自己的类（class）、对象（objects）和方法（methods）。掌握这些技能会让你制作更广泛的应用程序。

方法（Methods）

方法（Methods）是为了完成一个具体目标而组织在一起的一系列步骤。方法（Methods）和函数（functions）这两个概念非常相似。方法（Methods）是在一个类或者一个对象中，为完成一个目标组合在一起的一组步骤，而函数（functions）是为完成一个目标组合在一起的一组步骤，是独立存在的。这两个概念非常相似，成了同义词。我会经常听到人们在使用这两个单词时常常互换使用，甚至是他们想说的是一个方法，却使用了function函数这个单词。 能够编写自己的方法可以非常方便地代替一些重复的工作。如果你想写一个每天迎接造成的方法，会是这样子：

```
func goodMorning ( ) {
    println("Good Morning")
}
```

首先用func作为方法的开头，func用来声明新方法的开头，接着就是方法的名字，使用驼峰命名法命名。例如： `goodMorning( )`

> Page 97

方法名字后面跟着一对括号，接着就是左大括号，表示一个方法的开始，最后是右大括号，表示一个方法的结束。 为了能够提供一个更详细的决策过程，方法（Methods）可以接收外来的输入值（就是参数值）。例如，你要训练一个田径队，如果能够告诉你的运动员他们需要跑具体多远，可能会比较有帮助（跑步或者跑7英里）。

外来是输入值会改变最终的结果。这些外来的输入值叫做parameters（参数）。参数是外来的值，被传入到方法中。参数的名字和类型写在括号内，传入的参数的值也就是这些括号内的变量名的值。例如：

```
func goodMorning(name: String){
    println("Good Morning \(name)")
}
//Good Morning NAME
```

先写参数名字，然后是冒号，冒号后面就是参数的类型，如果方法有多个参数，用逗号间隔每个参数：

```
func goodMorning(name: String, day: String){
    println("Good Morning \(name), Happy \(day)")
}
 //Good Morning NAME-VARIABLE-HERE, happy DAY-VARIABLE-HERE
```

在你的方法中，你可以想写多少代码就写多少代码，不过基于最佳实践原则，最好把相似功能的代码写成一个方法，然后使用时再调用它。把你的代码变成多个可重复使用的小方法，这个好习惯，会让你的更新代码时轻松不少。

你刚刚编写了 `goodMorning` 方法，但是如何使用这个方法呢？调用（calling）方法是指，一行可以触发方法执行的代码。调用方法非常简单，写下方法的名字后跟一对括号。及时你的括号里并没有参数，这对括号也要写上。例如，这个名字为 `goodNight` 的方法：

```
func goodNight () {
    println("Good Night")
}
```

调用这个方法的代码是这样的：

```
goodNight()
// "Good Night"
```

> Page 98 | Chapter 4 : Diving Deeper

这样就能调用 `goodNight` 了，执行goodNight里的步骤。但是如果你的方法有参数值，像是 `goodMorning` ，如果调用呢？有参数值的方法一般都长这样：

```
func goodMorning(name: String, day: String){
    println("Good Morning \(name), Happy \(day)")
}
```

//Good Morning NAME-VARIABLE-HERE, happy DAY-VARIABLE-HERE

调用 `goodMorning` 方法的代码是这样的：

```
goodMorning("Steve", day: "Saturday")
//Good Morning Steve, Happy Saturday
```

最后，如果你的方法有参数，但是调用的时候忘记提供参数了，你可以为这些参数设置默认值。如果在调用时没有提供参数，就会使用默认值作为执行时的变量值。是这样设置默认值的：首先在类型后面写一个等号，然后等号右边写上这个默认值。例如，下面的代码把 `name` 和 `day` 的默认值设置成“World”和“Day”：

```
func goodMorning(name: String = "World", day: String = "Day") {
     println("Good Morning \(name), Happy \(day)")
}
```

现在，调用时忘记提供参数值，方法中打印出的信息仍然是可读的：

```
goodMoring()
// Good Morning World, Happy Day
```

返回值（Return Values）

除了能够接收输入值，执行一系列步骤，方法（Methods）还能做更多事情，还能产生输出值。当一个方法提供了一个输出值，这个输出值叫做返回值（return value）。返回值是方法的最终结果，返回给调用者。一个返回值可以是任何类型的，不过返回值的类型必须在方法中声明：

```
func sum(a: Int,b: Int) -> Int {
     return a + b
 }
```

如果一个方法有返回值，必须在参数后面定义一下。返回类型（return type）是被返回来的变量的类型。为了定义返回值的类型，首先要写一个dash（英文输入法下的破折号），然后一个大于号，看起来像一个剪头，表示这个方法有返回值，剪头的右面是返回值的类型。最后，`return` 关键词还要出现在方法中，`return` 后面是要返回的变量。

> Methods | Page 99

不管方法中的代码有多少行，`return` 表示这个方法结束了，编译器在编译方法中的代码时，遇到 `return` 就结束编译，`return` 后面的代码就会被忽略掉：

```
func calculateTip(bill: Float, percentage: Float)-> Float {
    var tip = bill * percentage
    return tip
 }
```

类（Classes）

假设你是一个建筑师，你刚刚签了一个合同，要在一个新的小区修建20个相似的房子。在你派出建筑工队之前，你必须要画一个房子的设计图。这份设计图将会展现房子的外表和功能。把这份设计图当做模板，就能制作出20个房子各自的设计图了。 使用设计图或者模板来建造物品能够节省时间，让后期维护工作更加简单。这个原则同样适用于创建虚拟对象。在Swift中起着设计图功能的叫做类（class）。类是一个对象（object）的设计图或模板，这个模板可以反复使用来创建虚拟对象。

一个类定义了对象的属性和行为。 属性就是对象具有的特征。属性（attributes）由properties定义，(properties也有属性的意思，所以我就用英文了，不知道该怎么翻译才不会有歧义啊），properties就是对象中的变量。一个房子的properties可能是外部颜色，街道号码，浴室的数量。 行为就是对象提供的方法。例如，方法可能是，把空调温度调整到68华氏度，打开车库的门，凯琪报警系统。 在Swift中创建你自己的类是非常简单的。如果你想为你的新房屋建设项目写一个类，它看起来会是这样：

```
class House {

 }
```

类的名字的开头是大写英文字母，这样能让其他的开发者区分出这时一个类还是一个对象。对象和方法常常用一个小写字母开头。

属性（Properties）

给你自定义的类增加属性和声明一个变量非常相似。你可以给你的 `house class` 增加一些属性：

```
class House {
  var exteriorColor = "Brown"
  var currentAirTemp = 70
  var isGarageOpen = true
  var isAlarmEnabled = false
}
```

> Page 100 | Chapter 4 : Diving Deeper

你给类增加了4个属性，每个都设置了默认值。这意味着参数没有传入新值时，从这个类中建造的房子都会有一个褐色的外表，车库的门是开启的，报警系统是关闭的，空调设置的温度是70华氏度。

方法（Methods）

你也可以给你的类增加一些方法：

```
class House {

  var exteriorColor = "Brown"
  var currentAirTemp = 70
  var isGarageOpen = false
  var isAlarmEnabled = false

  func prepareForNighttime(){
      isGarageOpen = false
      isAlarmEnabled = true
      currentAirTemp = 65
   }

  func prepareForDaytime(){
      isGarageOpen = true
      isAlarmEnabled = false
      currentAirTemp = 70
    }

  }
```

创建一个对象（Creating an Object）

现在你已经为你的类（house class）定义了属性和方法，是时候来建造第一个房子了。为了从一个类中创建一个对象，你需要调用初始化方法（initializer method）。这个初始化方法是专门为了设置新对象而设计的。初始化方法有些像你第一次打开Mac时遇见的安装向导。

> Creating an Object | Page 101

如果你的参数有默认值，那么就没有必要写初始化方法了，Xcode会为你创建一个。初始化方法是这样写的，先写一个类的名字，然后跟上一对括号，例如：`var myHouse: House = House()` 或者也可写成这样： `var myHouse = House()` 不管那种方法都可以创建一个新的房子叫做 `myHouse` 。

获取属性（Accessing Properties）

用点语法可以获取属性的值，点语法是可以取值或赋值的格式。点语法始于一个对象：`var myHouse = House()` 为了获取 `exteriorColor` ，先写变量名字，后面跟一个点，然后是 `exteriorColor` ：

```
myHouse.exteriorColor
//Brown
```

`exeriorColor` 返回是字符串"Brown"，因为这个属性的默认值是"Brown"。 这个简单的格式也可以用于给属性赋值，为了赋值，需要增加一个等号，然后写上新的值：`myHouse.exteriorColor = "Blue"` 这样就把 `exeriorColor` 的值改成了"Blue"：

```
myHouse.exteriorColor = "Blue"
 println(myHouse.exteriorColor)
//"Blue"
```

当代码把 `exeriorColor` 的值改成"Blue"后，接下来的一行打印出了房子的 `exeriorColor` 值。

> Page 102 | Chapter 4 : Diving Deeper

调用方法（**Calling Methods**）

调用house类中的方法直接写上类的方法名字即可： `myHouse.prepareForNighttime()` 调用方法和之前你看到调用属性都是使用点方法。首先，你先写一个对象的名字，然后一个点，跟着你想调用的方法的名字，最后是一对括号。如果这个方法有参数，那么写上参数值，如果没有参数，写一个空着的括号即可。

子类（**Subclasses**）

当你在创建什么东西的时候，重复使用模板会比重新画草稿好的多。假设你想建造一个木屋，木屋和房子有很多的相似之处，不过也有一些不同的属性和行为。这时，你可以借用house类然后做一些改变，而不是去重现画一个设计图。这个重新使用的过程就叫做子类化（subclassing）。 子类化能够使子类和父类之间简便地分享属性和行为。在这个例子中，父类是 `House` ，而子类是 `Cabin` 。写一个子类和创建一个类差不多，不同的是先有子类的名字，然后一个冒号，父类的名字是在冒号之后。例如：

```
class Cabin: House {

}
```

继承（**Inheritance**）

继承是指的父类的属性和行为能够传递给子类。所有父类的属性和行为都会自动地传递给子类。例如，cabin类继承了 `exteriorColor` ， `currentAirTemp` ， `isGarageOpen` ，和 `isAlarmEnabled` 的属性以及属性的默认值。cabin类还继承了 `prepareForNighttime()` 和 `prepareForDaytime()` 两个方法。然而，如果有需要，cabin类可以增加或者修改这些属性值。例如，cabin类可能想把 `exteriorColor` 的默认值设置成"Red"。

> Subclasses | Page 103

重构重写（**Overriding**）

如果你不想要一个褐色的小木屋怎么办？没关系，有了重写（overriding）就可以实现了。在子类中改变继承来属性或方法就叫做重写。重写使子类可以用用自己的特定的属性和方法。为了重写一个方法，我们会在方法前面增加一个override关键词。

初始化重写（**Overriding initializers**）

你可以用初始化方法来重写属性的默认值。无论什么时候，创建一个新的对象后，都要调用初始化方法，这意味着你要写你的初始化方法和默认值：

```
class House {
    var exteriorColor = "Brown"
    var currentAirTemp = 70
    var isGarageOpen = false
    var isAlarmEnabled = false

    func prepareForNighttime(){
    isGarageOpen = false
    isAlarmEnabled = true
    currentAirTemp = 65
  }

  func prepareForDaytime(){
    isGarageOpen = true
    isAlarmEnabled = false
    currentAirTemp = 70
  }
}

class Cabin: House {
    override init(){
        super.init()
        exteriorColor = "Red"
    }
}
```

cabin类用init关键词定义了自己的初始化方法。因为cabin类是重写的House类的初始化方法，所以需要在方法开头加上override关键词。记住，House类的初始化方法是自动创建的。init关键词后面跟着一对空括号，然后是左大括号，后面写初始化方法的具体内容。 在有子类的情况下，在重现或者修改子类之前，父类的初始化方法要先被调用。就父类而言，使用super关键词。在子类的init方法中，写 `super.init()` 。这就能够调用父类的初始化方法了。一旦完成初始化，子类就可以自由地去重写和修改了。在这个例子中，cabin类把 `exteriorColor` 修改为"Red"。

> page 104 | Chapter 4 : Diving Deeper

重写方法

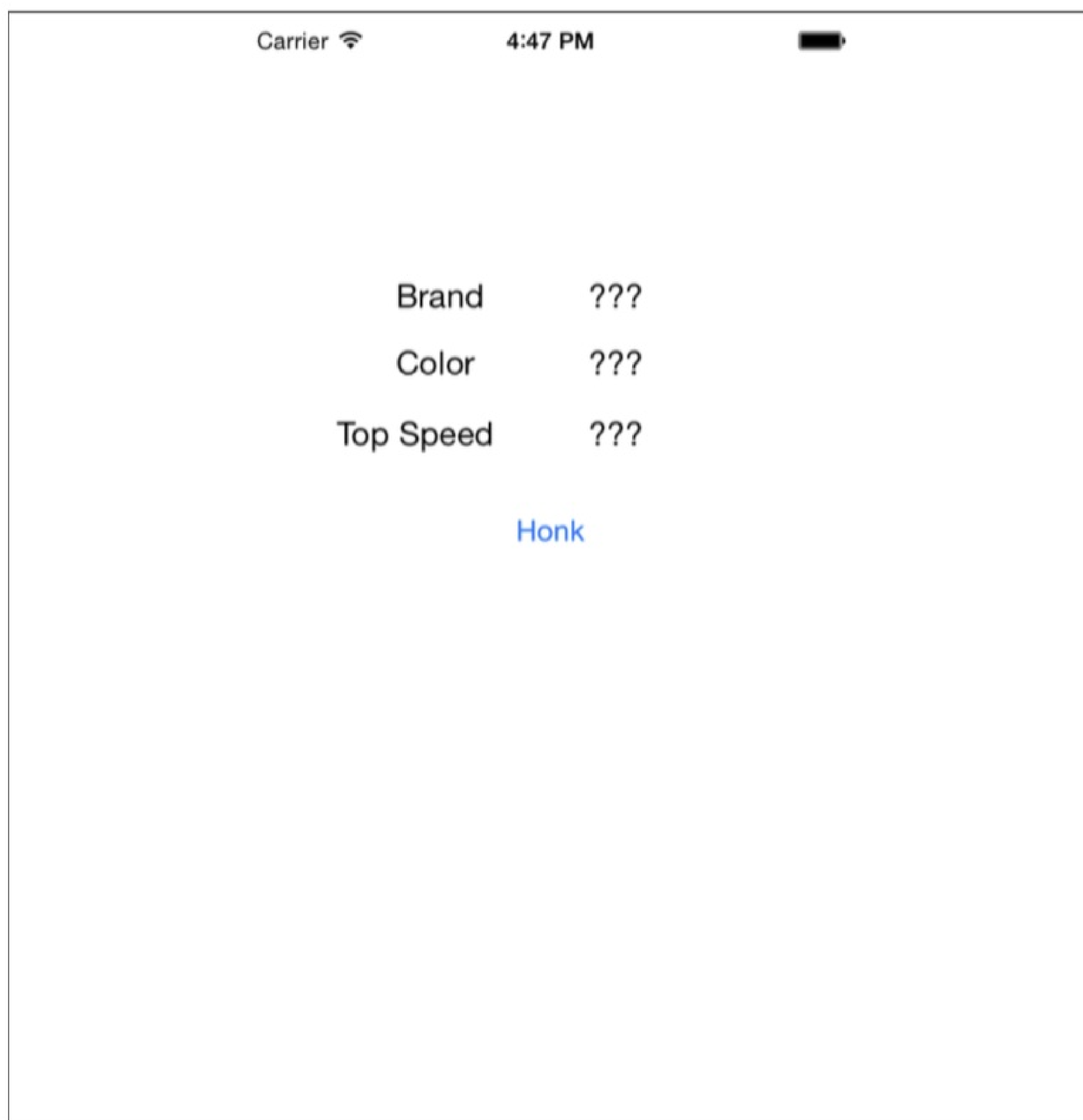
除了能够重写子类的属性之外，你还可以重写子类的方法。例如，当你为cabin设置nighttime时，就和House类不太相同。在一个House里，你可能想关上车库的门，改变空调的温度，开启报警系统，然而，在一个cabin里，没有报警系统，也没有车库，你更想在壁炉中生火。重新方法需要用到override关键词：

```
class Cabin: House {
    override init(){
        super.init()
        exteriorColor = "Red"
}
    override func prepareForNighttime(){
        startFire()
    }
}
```

在这一章中，你了解了更多swift知识，学习了如何创建自己的类和方法，如何创建子类和重写，有了这些技能，你已经能够编写任何你需要的类了。你的知识正在变多，现在是时候把这些知识应用到实践中去了，继续我们的Race Car练习吧。

第四章练习Race Car - 编写一个跑车App吧

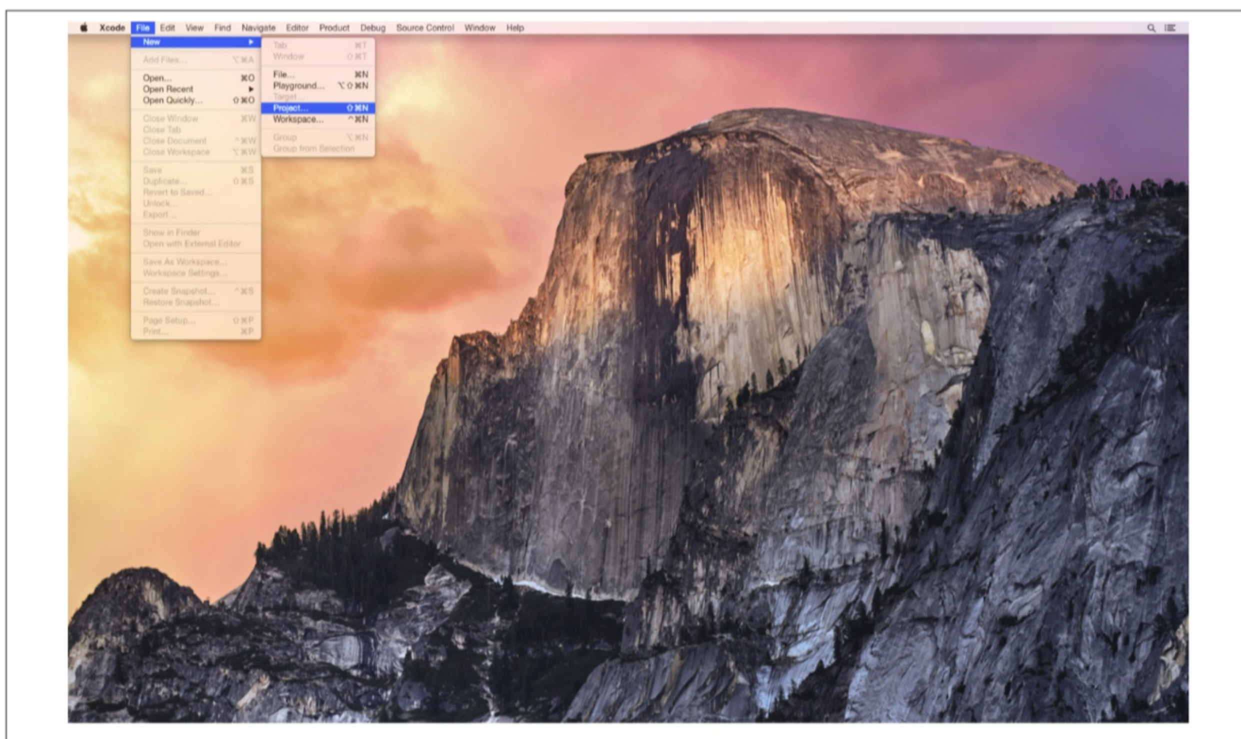
在这个练习中，你将创建race car类，App会提供一辆赛车，点击Honk（按汽车喇叭），就会展示这赛车的详细信息。App的用户界面请见图4-1。



Exercise: Race Car | Page 105

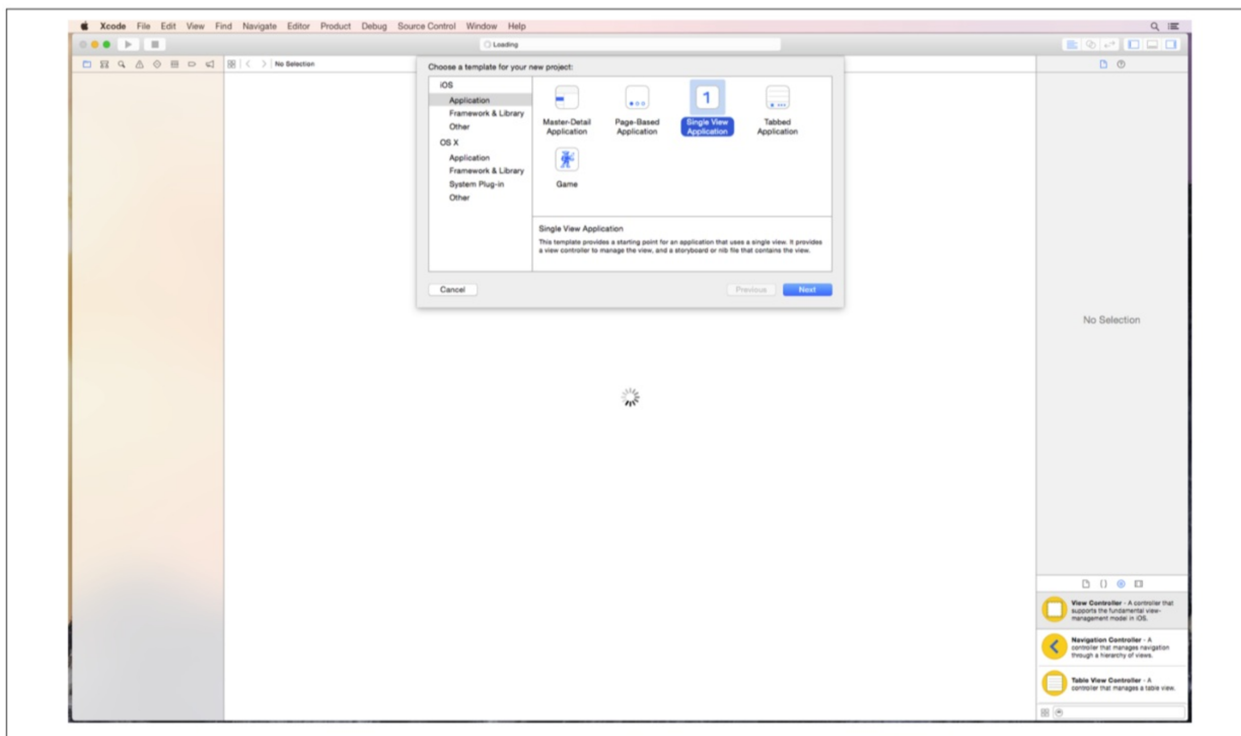
界面上有六个Label和一个Button控件，App可以显示赛车的属性，用户可以点击Honk，点击Honk时，不会响起汽车喇叭声，而是会在开发者日志（developer log）中写下一段信息。

你已经知道了App有哪些功能，那么打开Xcode吧，点击顶部菜单栏的File -> New -> Project（见图4-2）。

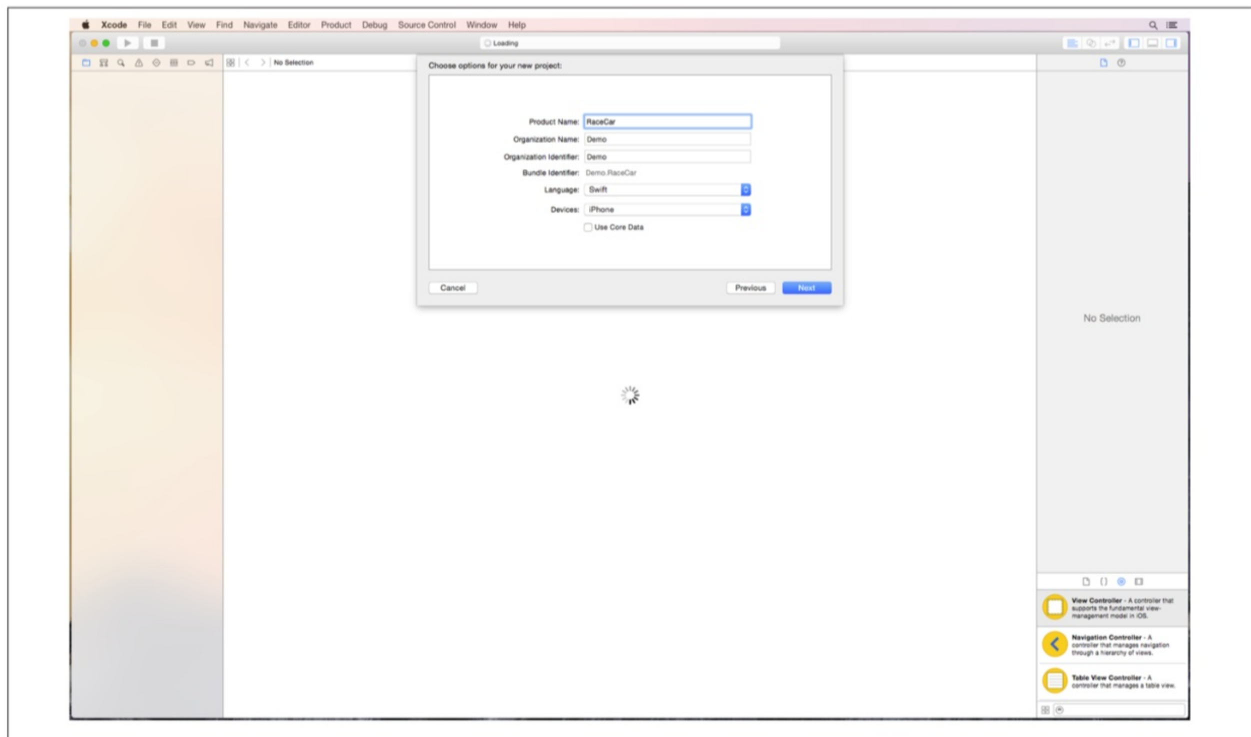


Page 106 | Chapter 4 : Diving Deeper

从模板中选择Single View Application（见图4-3）。



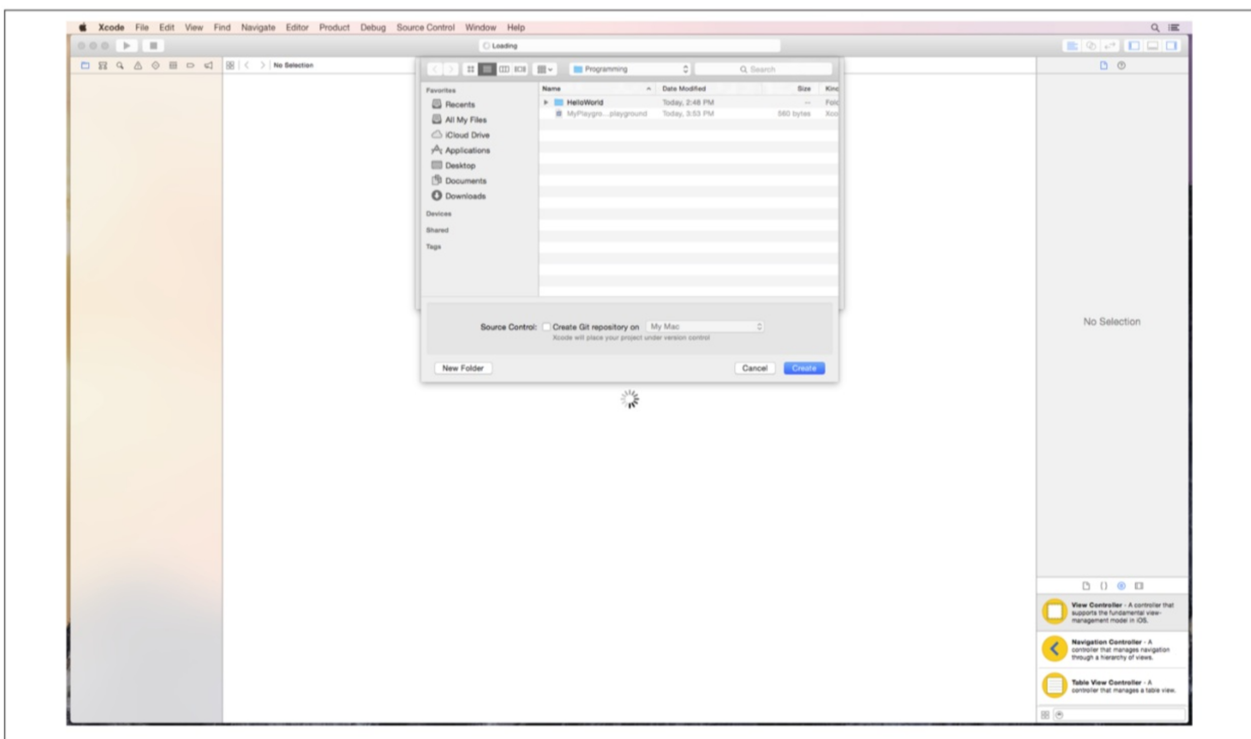
点击Next，在Product Name一栏输入RaceCar（见图4-4）



Exercise: Race Car | Page 107

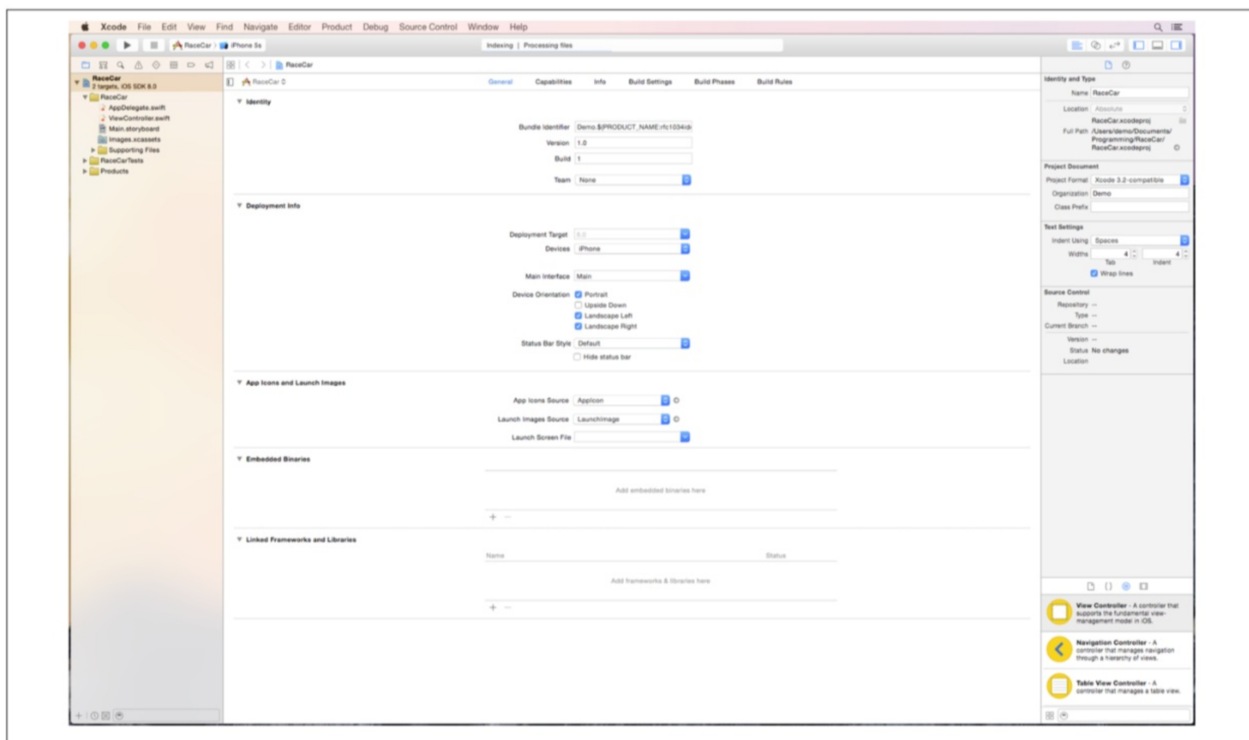
Organization Name和Organization Identifier已经自动填写好了，如果没有填写好，输入你的姓名中间不要有空格。最后，Language选择Swift，Devices选择iPhone，点击Next。

接下来从左侧选择你要存放的文件夹，点击Create，保存工程（见图4-5）。



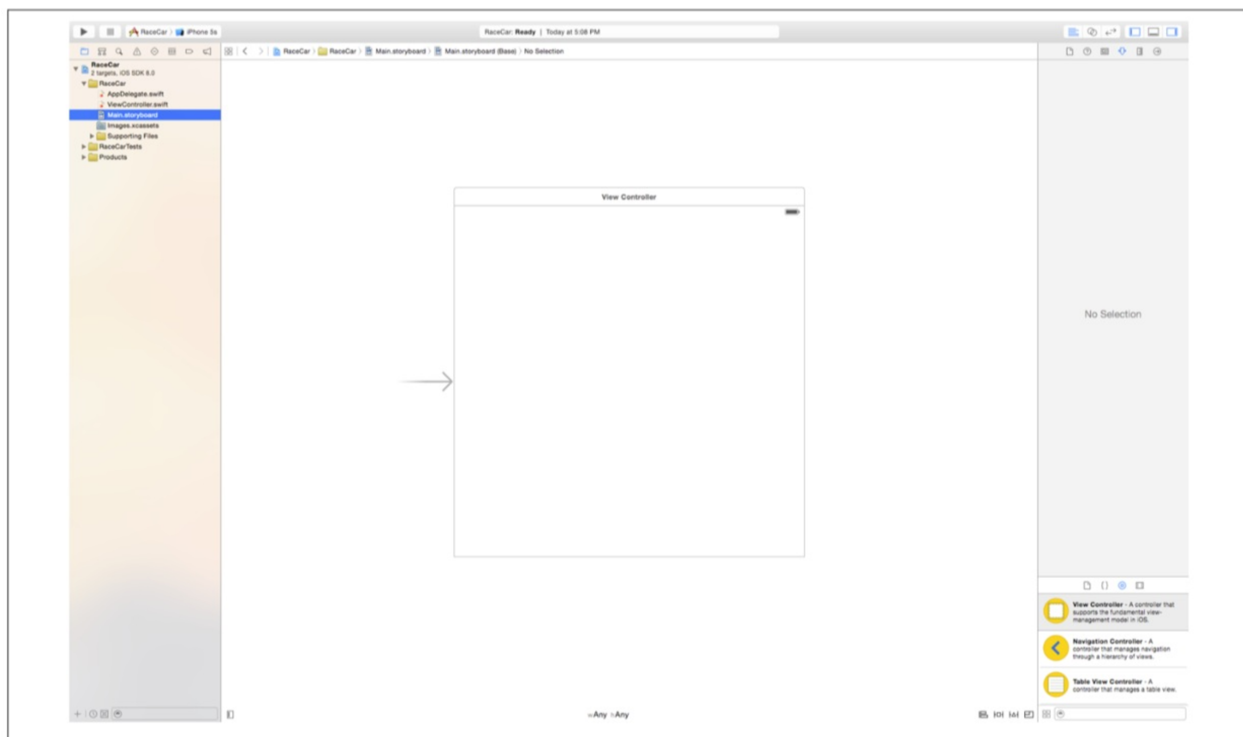
Page 108 | Chapter 4 : Diving Deeper

出现了工程的详细信息界面（见图4-6）。Project Navigator在左边，Standard Editor在中间，Inspector在右边。



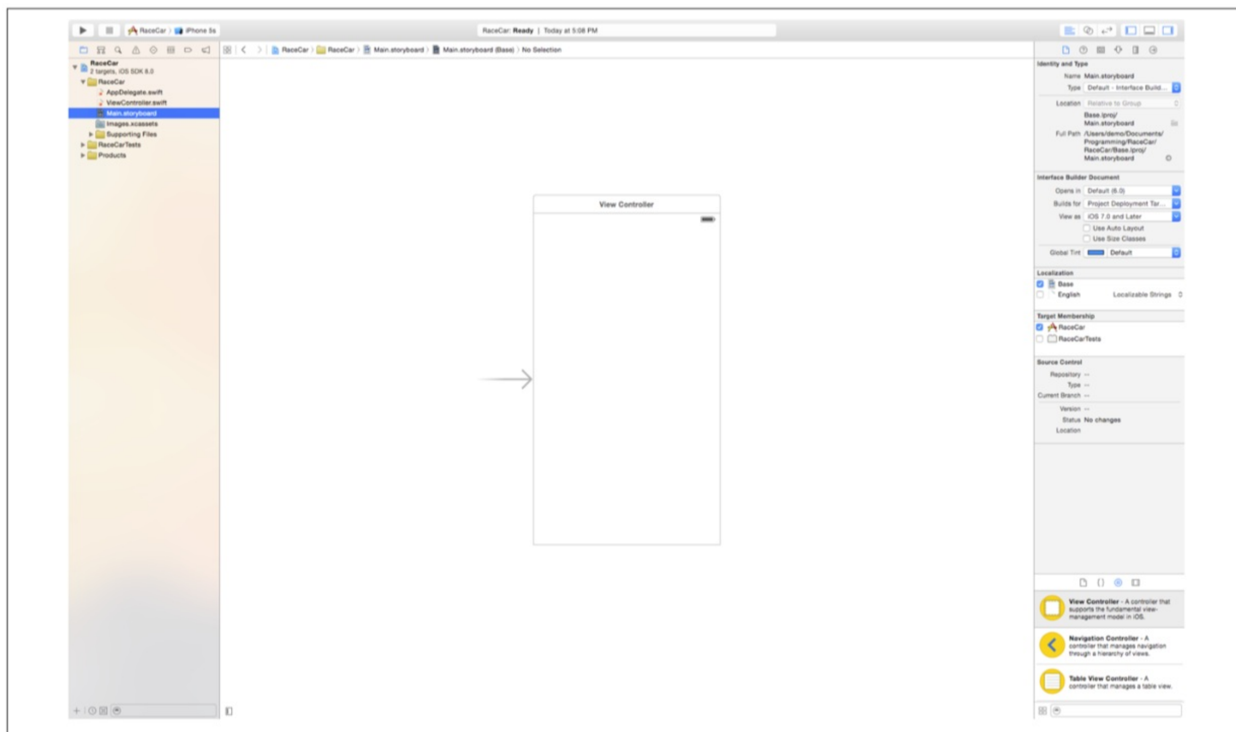
> Exercise: Race Car | Page 109

点击Project Navigator中的Main.storyboard文件，出现一个空白界面（见图4-7）。



我们这次开发的App仅限于在iPhone上使用，点击Inspector的上方第一个按钮，看起来像是一张纸折了一个角。到中间部分，不勾选Use Auto Layout选项。这时会出现一个对话框，选择iPhone。然后不勾选Disable Size Classes。这时，Storyboard中的界面形状会改变（见图

4-8）。我们将会在第七章详细介绍Auto Layout的知识。



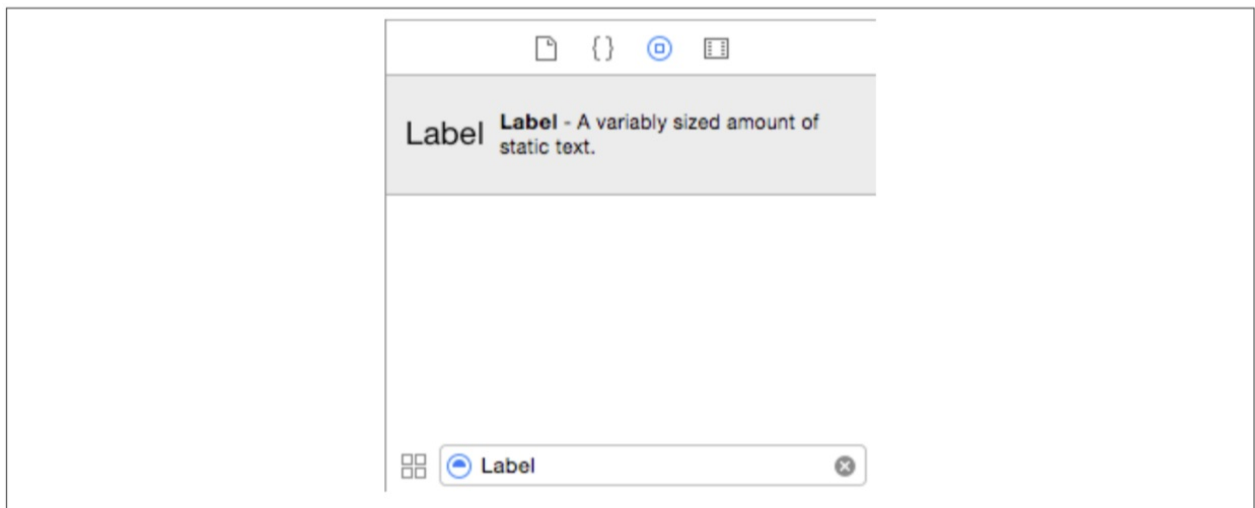
> Page 110 | Chapter 4 : Diving Deeper

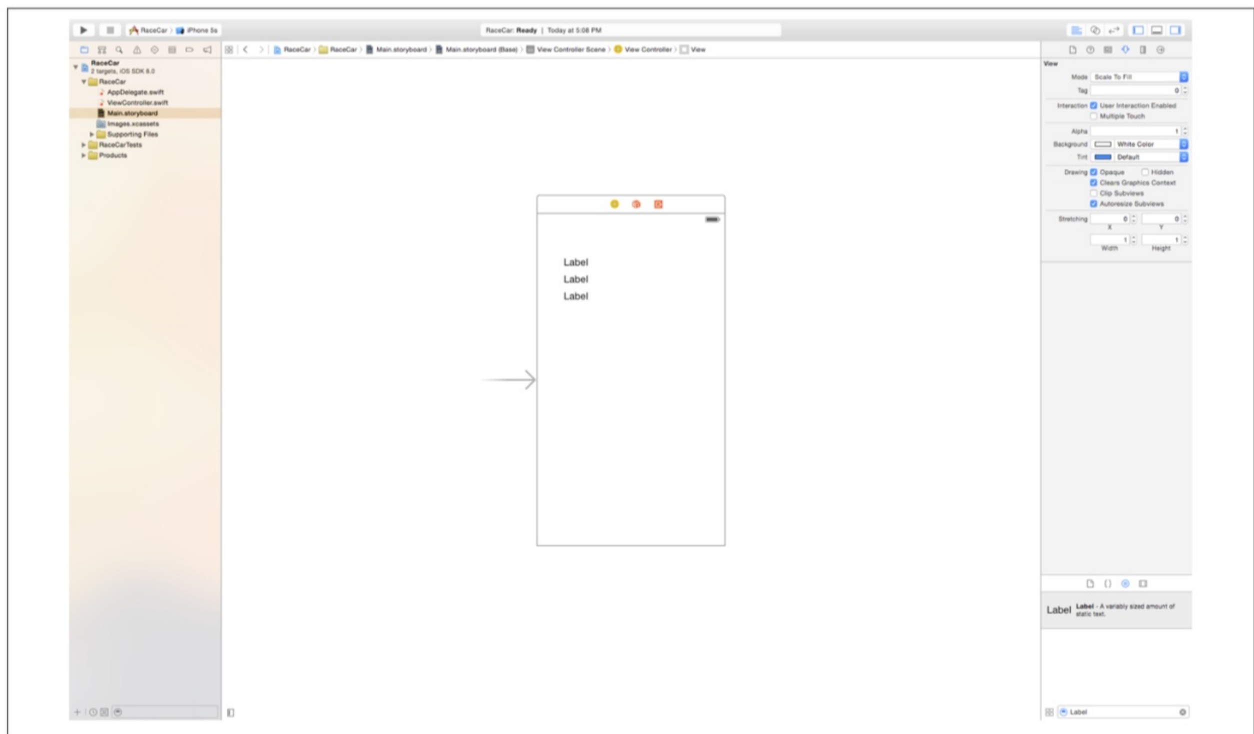
确保Inspector在屏幕的右方。如果没有出现在屏幕右方，点击Inspector View Button（见图4-9），就是右上角右边有一条条纹的按钮，这样就可以显示Inspector了。



Inspector下方有个小的工具栏按钮，Object Library图标是从左往右第三个（圆圈中有个小方块）。

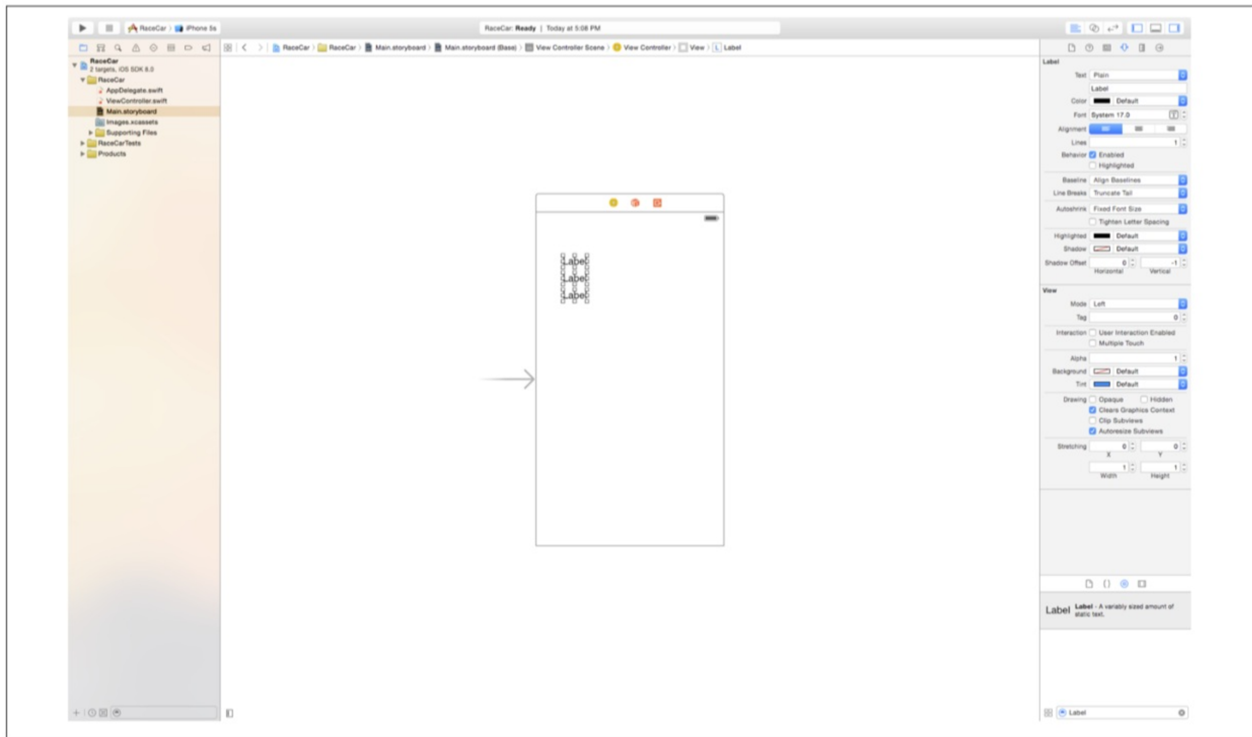
打开Object Library后，在搜索输入Label（见图4-10），将三个Label控件拖入到界面中，垂直对齐放在界面的右边（见图4-11）。

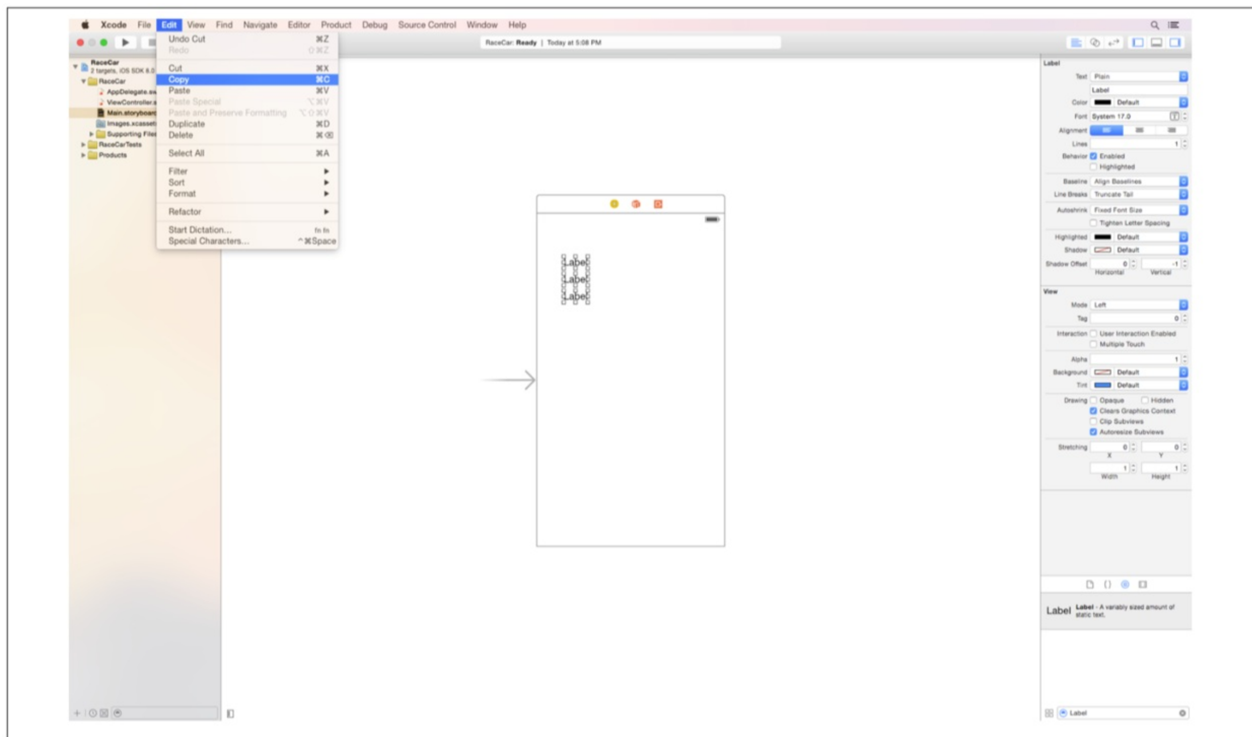




Exercise: Race Car | Page 111

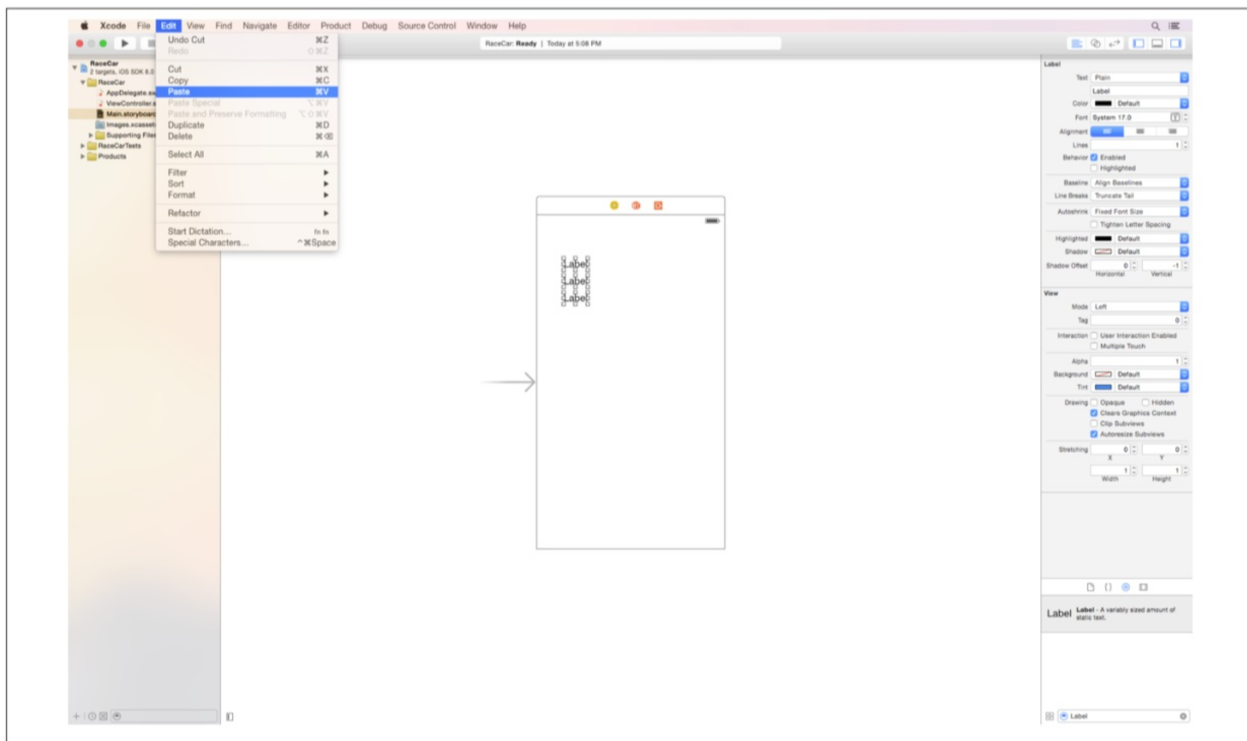
三个Label对齐后，按住鼠标左键然后拖拽，覆盖三个Label，这样三个Label就被选中了（见图4-12），接着，到顶部菜单栏中选择Edit -> Copy或者按Command+C（见图4-13）。





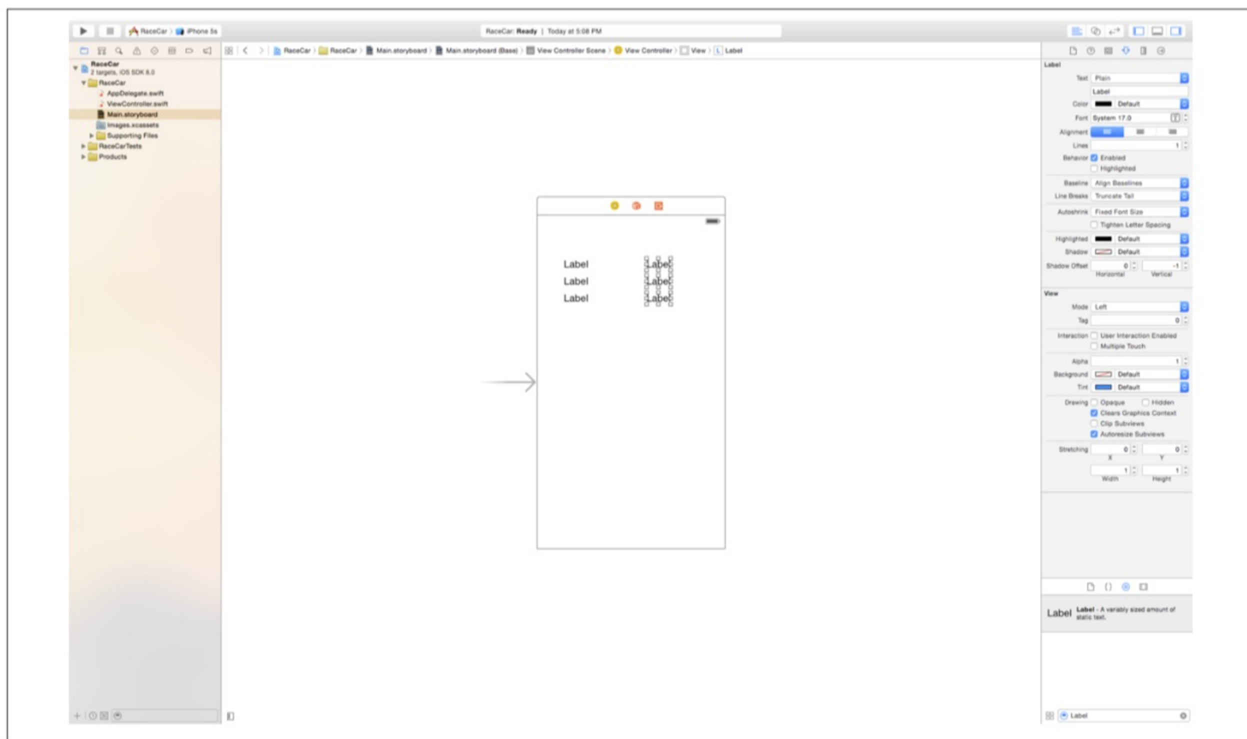
Page 112 | Chapter 4 : Diving Deeper

接着到顶部菜单栏选择Edit -> Paste或者按Command+V（见图4-14）。



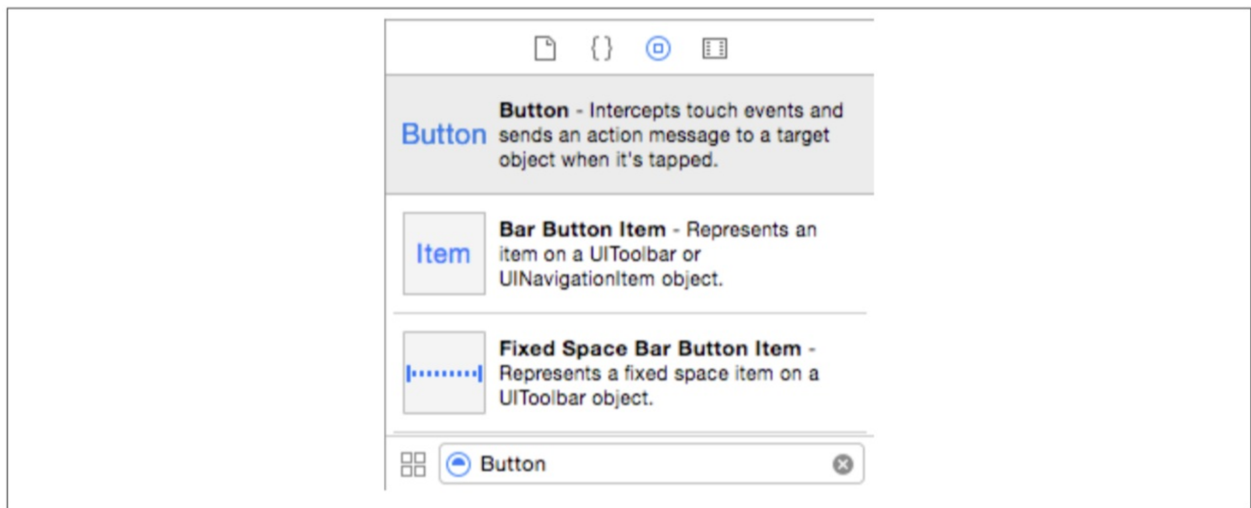
Exercise: Race Car | Page 113

这样又添加了三个Label控件，把这三个新的Label选中，然后拖到界面的右边，让Label对齐（见图4-15）。

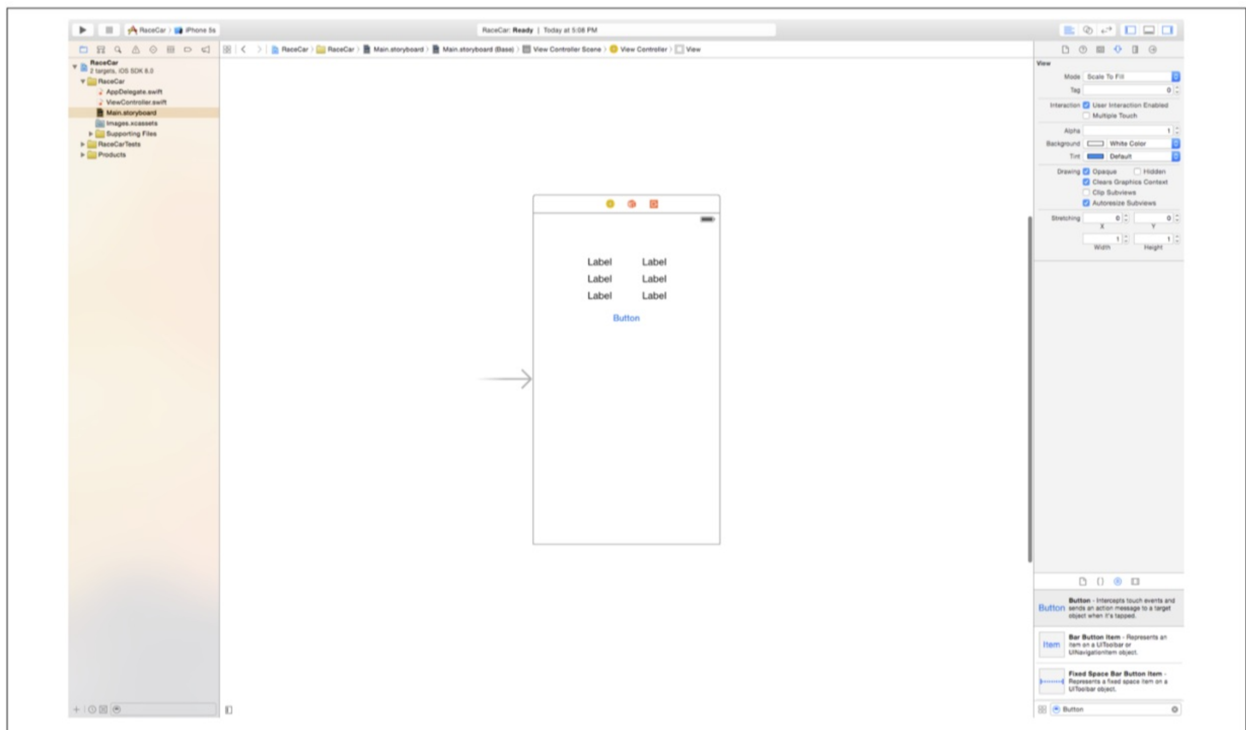


Page 114 | Chapter 4 : Diving Deeper

有时候复制粘贴控件比起从Object Library中拖拽控件会更方便。最后，把Object Library搜索框中的label文字删掉，输入button（见图4-16）。

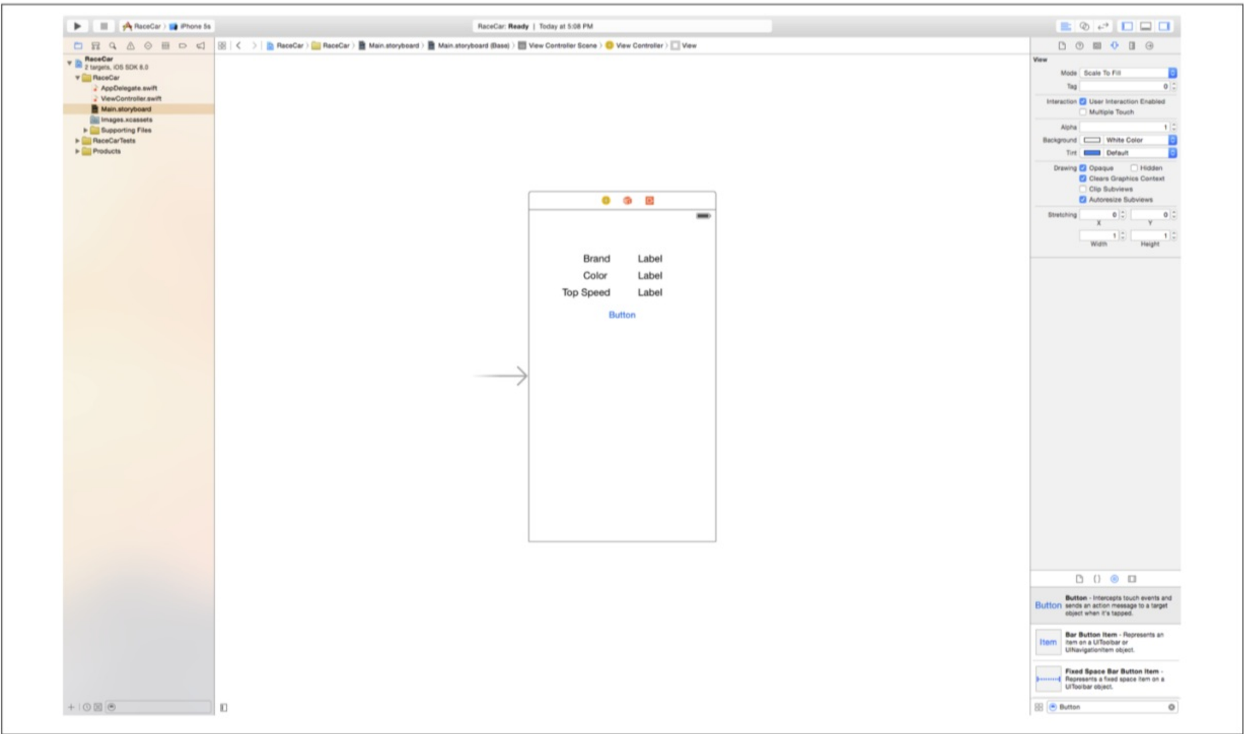


从Object Library中拖一个Button控件，放到两组Label中间（见图4-17）。



Exercise: Race Car | Page 115

双击界面左上角的Label，替换文字为Brand，双击左边中间的Label，文字更换为Color，最后双击左边最下方的Label，文字更换为Top Speed（见图4-18）。如有需要，重新调整Label的对齐。

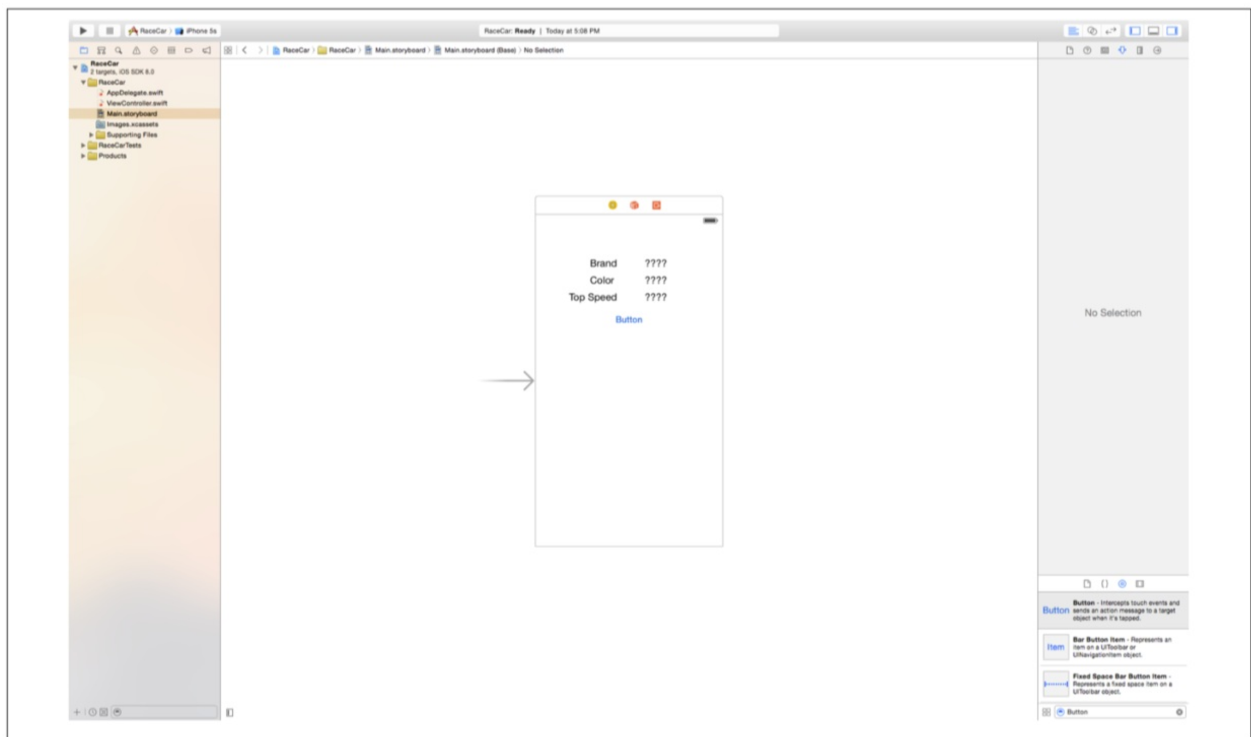


Page 116 | Chapter 4 : Diving Deeper

右边Label中展示左边Label的具体值。 见表格4-1。

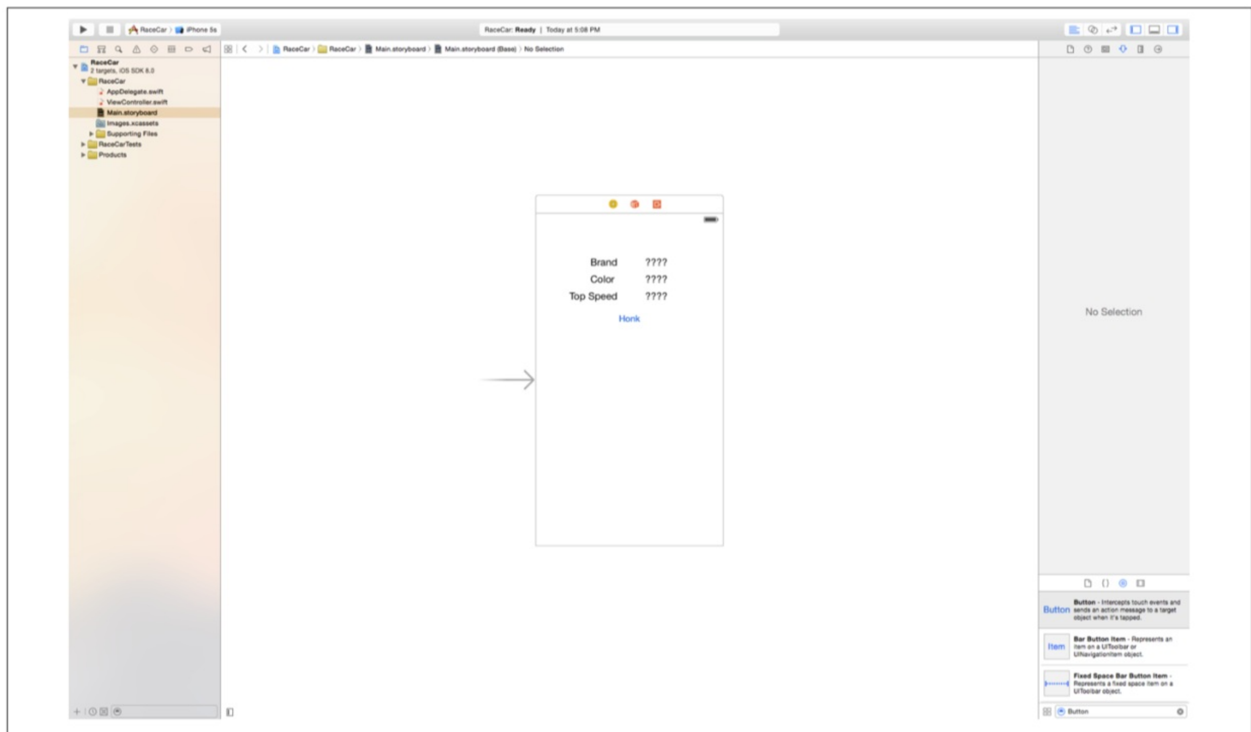
表格**4-1**	**Race Car**
品牌	Ferrari
颜色	红色
最高时速	200 mph

双击Storyboard中的右侧的Label，然后问text修改为????（见图4-19）。然后将右侧的Label宽度拖拽至60pts。



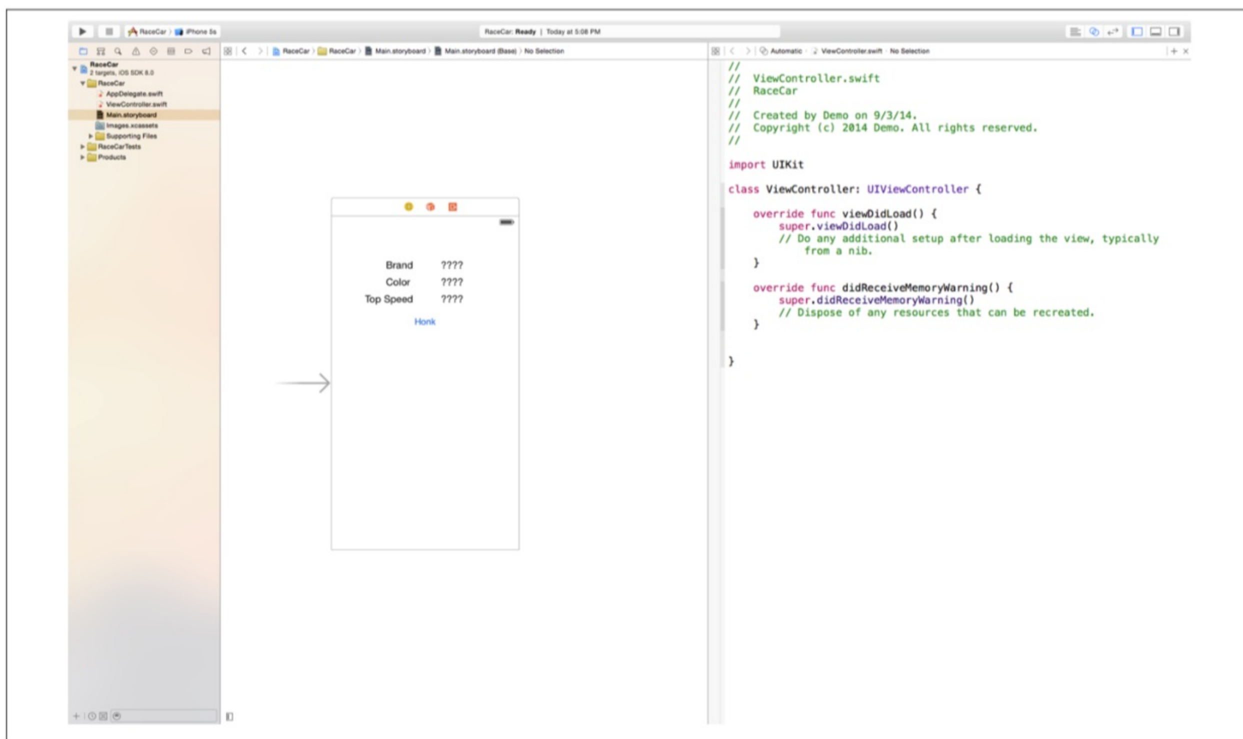
Exercise: Race Car | Page 117

最后，双击Button然后输入Honk，接着拖拽Button右上角的小方块，把宽度调整到60pts（见图4-20）。



Page 118 | Chapter 4 : Diving Deeper

现在，界面已经布局完成了，接下来需要把界面和controller连接起来。在这里，controller就是*ViewController.swift*文件。点击屏幕右上角的Assistant Editor按钮，打开Assistant Editor界面（见图4-21）。



为了能够有更多空间，我们把Inspector隐藏起来（点击屏幕右上角的Inspector View按钮）。

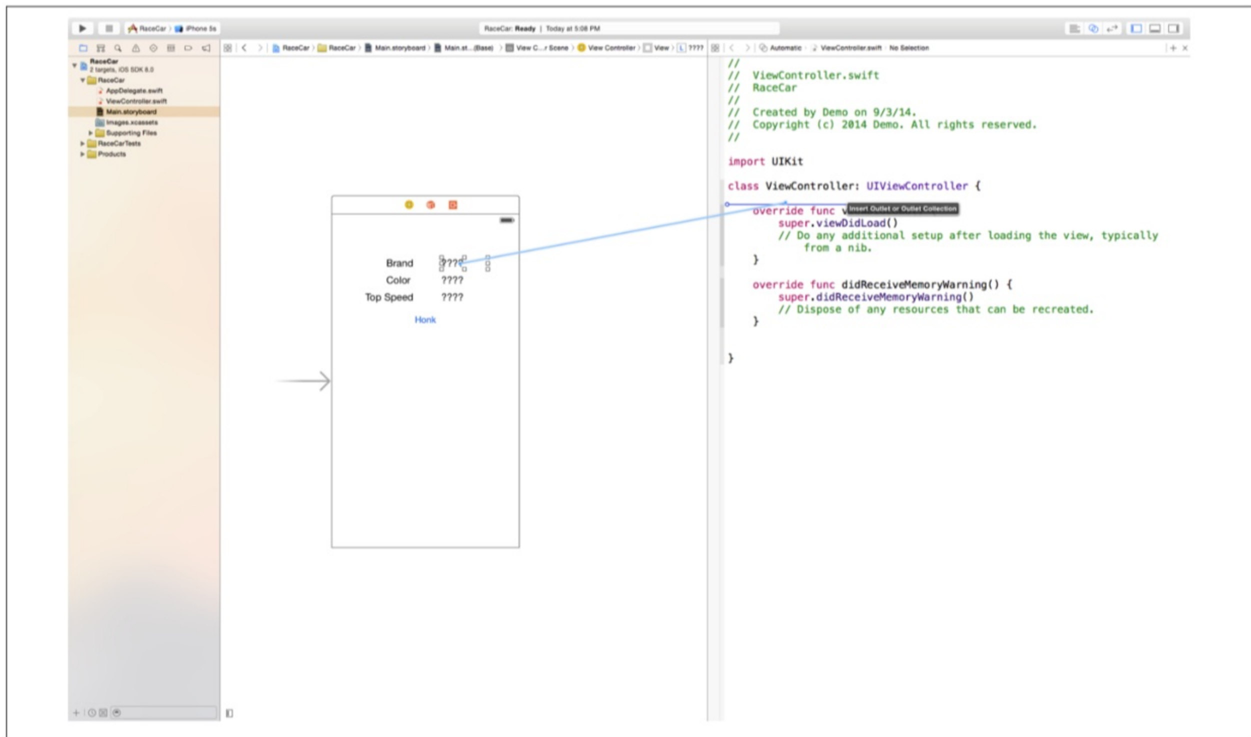
Assistant Editor会自动打开*ViewController.swift*文件，可以同上Assistant Editor顶部文件路径确认，就在Automatic这个词后面。

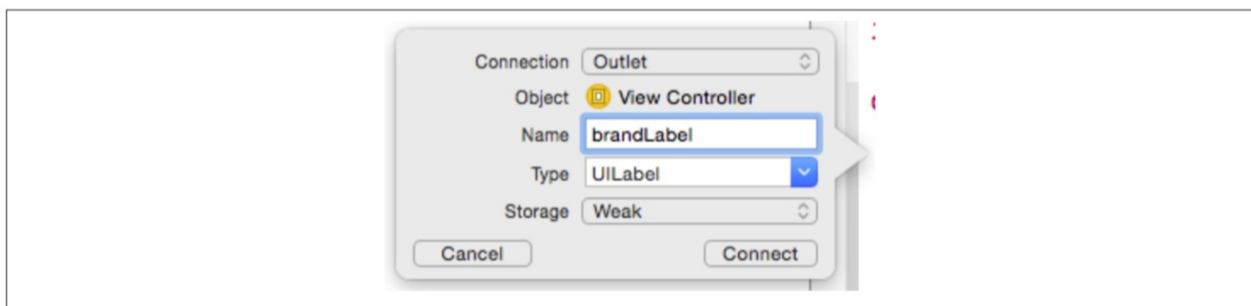
不是所有的控件都要连接到*ViewController.swift*文件中，如果这个控件是静态的或者无需变化，就没有必要连接。左侧的三个Label一直不变，那么就不需要连接。

选择右侧的Label，按住Control按钮，拖到下面这行代码的下方：

```
class ViewController: UIViewController {
```

当出现一条蓝色横线时，松开鼠标（见图4-22）；弹出连接对话框窗口（见图4-23）。





> Exercise: Race Car | Page 119

connection选择Outlet，Name输入brandLabel，点击Connect。

之后会自动生成一行代码：

```
@IBOutlet var brandLabel : UILabel!
```

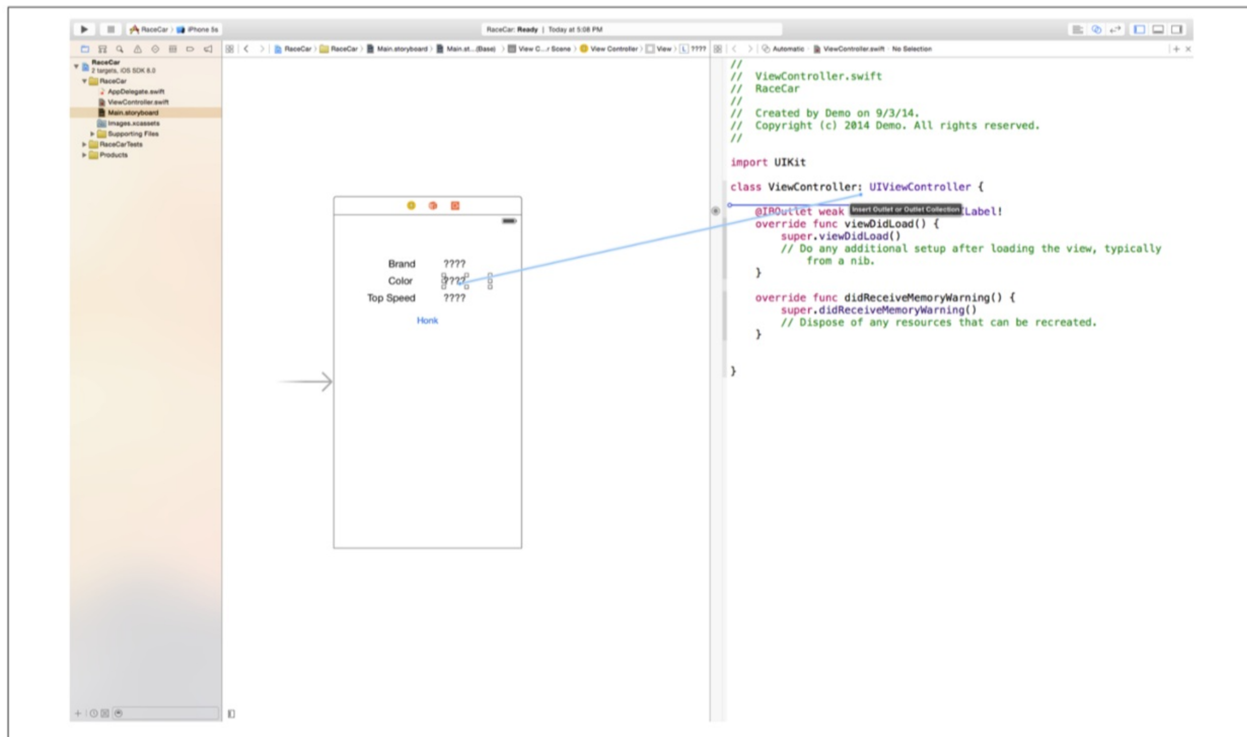
第一个关键词@IBOutlet是来表示这是一个oulet连接类型，var表示这是一个变量，brandLabel就是这个变量的名字，冒号后面跟着变量的类型UILabel。brandLabel变量就代表了右侧最上面的Label。

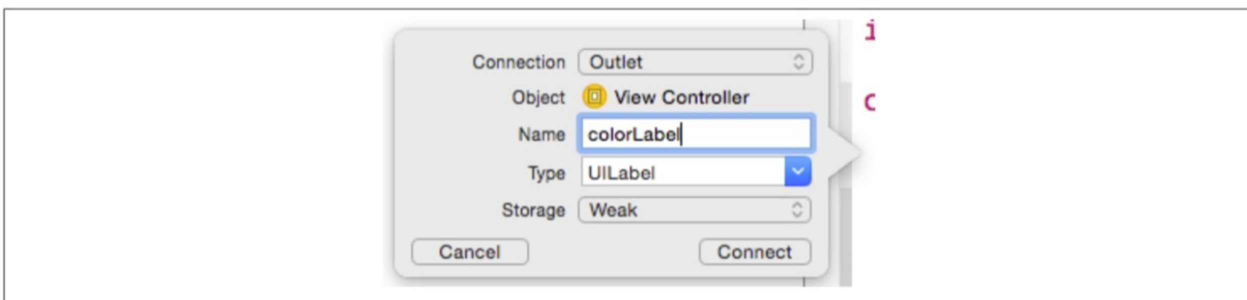
> Page 120 | Chapter 4 : Diving Deeper

然后选择界面上右侧中间的Label，同时按住Control键，拖到下面这行代码的下方：

```
class ViewController: UIViewController {
```

注意不要在brandLabel的代码上松开鼠标，如果你把鼠标错误的放到了其他的Label代码上方，和出现一个蓝色盒子。一定要在出现一条蓝色横线时，松开鼠标（见图4-24）；弹出连接对话框窗口（见图4-25）。





connection选择Outlet，Name输入colorLabel，点击Connect。

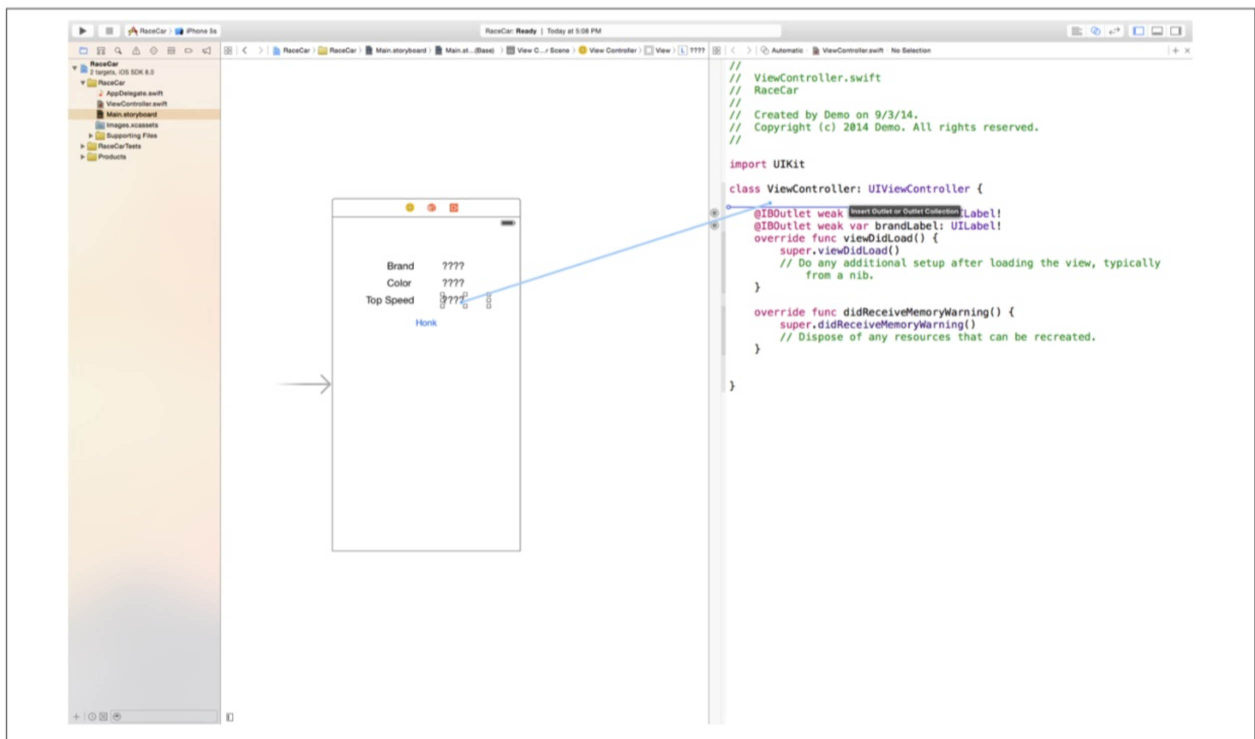
> Exercise: Race Car | Page 121

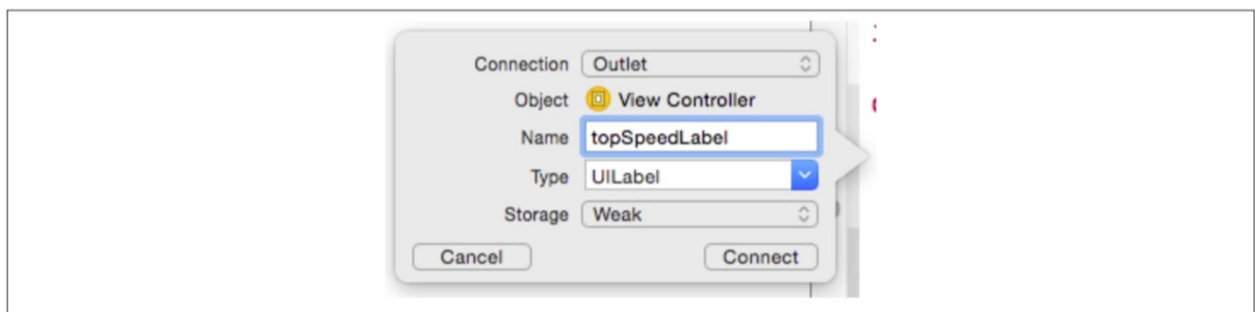
之后会自动生成一行代码。

然后选择界面上右侧底部的Label，同时按住Control键，拖到下面这行代码的下方：

```
class ViewController: UIViewController {
```

当出现一条蓝色横线时，松开鼠标（见图4-26）；弹出连接对话框窗口（见图4-27）。





connection选择Outlet，Name输入topSpeedLabel，注意使用驼峰命名法，点击Connect。

会自动生成一行代码。

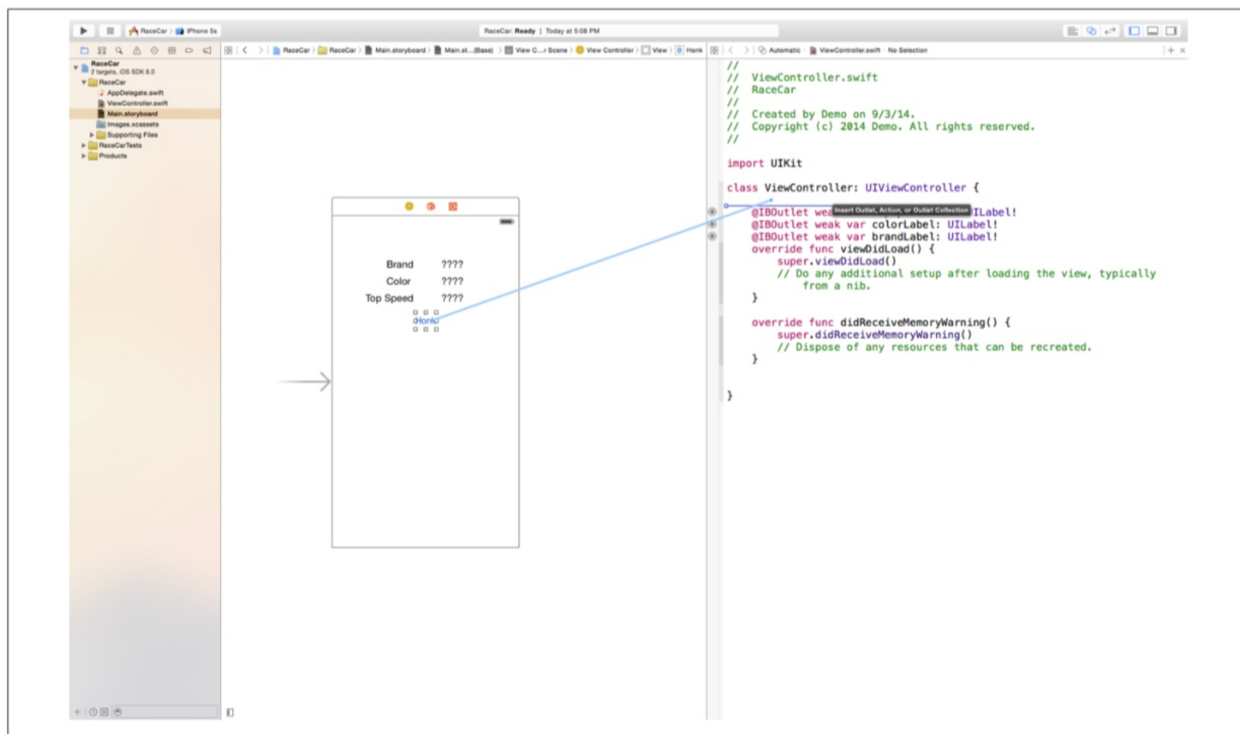
> Page 122 | Chapter 4 : Diving Deeper

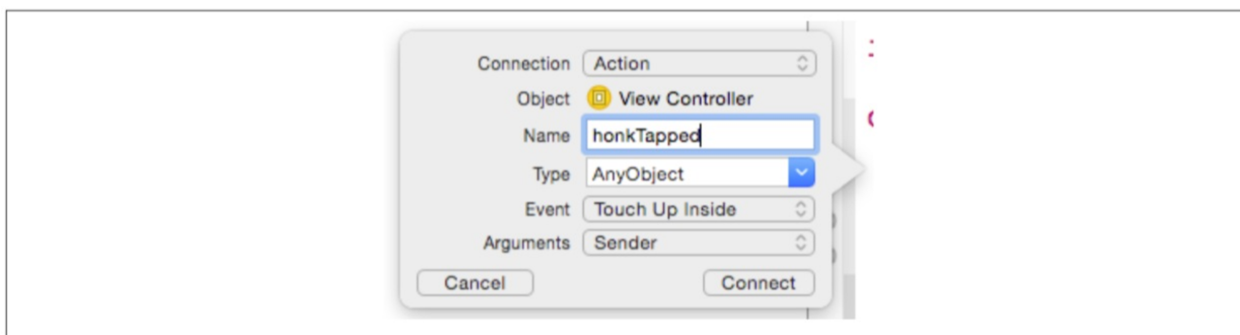
最后把Button连接到controller中。当用户点击Button时，Button会在controller中触发一个事件，这种连接模式叫做action。

选中界面中间的Button，同时按住Control键，拖到下面这行代码的下方：

```
class ViewController: UIViewController {
```

当出现一条蓝色横线时，松开鼠标（见图4-28）；弹出连接对话框窗口（见图4-29）。





Exercise: Race Car | Page 123

因为这次连接的控件是Button，所以connection选择Action，点击Connection右侧的下拉菜单，选择Action，然后Name输入honkTapped，点击Connect。

会自动生成一行代码，Action连接创建的是方法，Outlet连接创建的是变量。每次用户点击Button时，honkTapped方法就会被调用。看一下Xcode生成的这段代码：

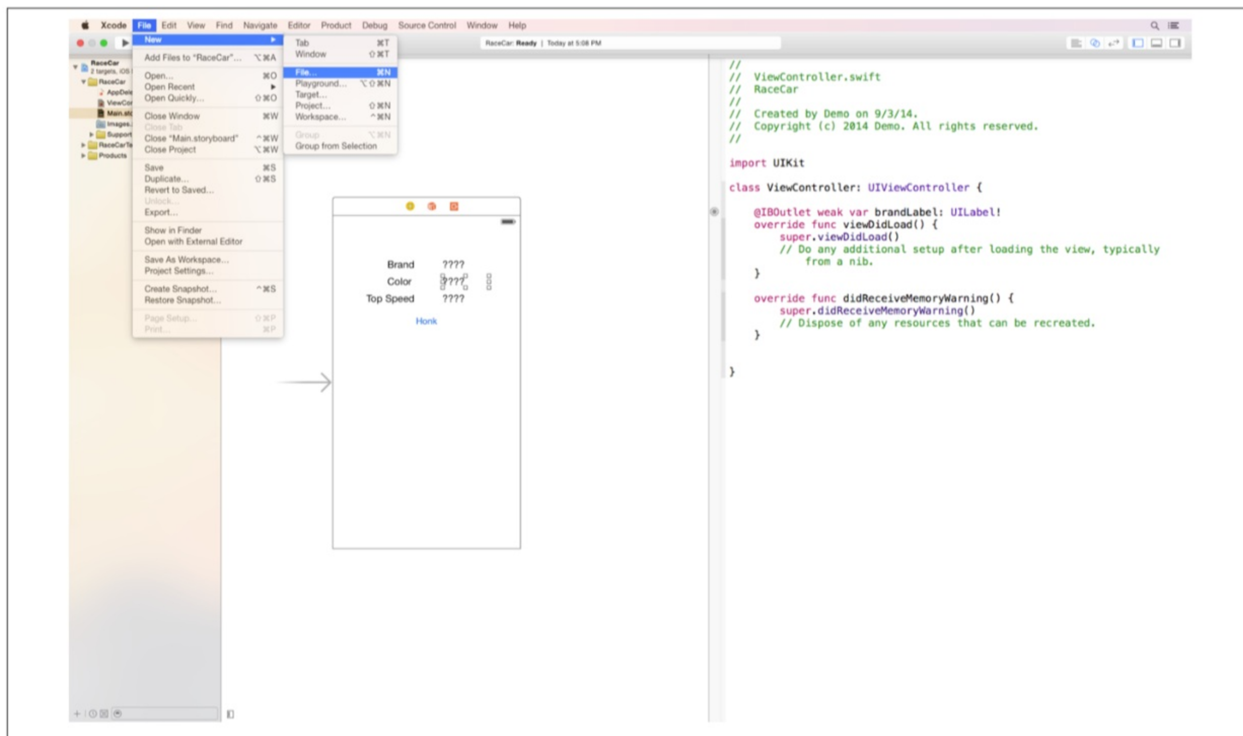
```
@IBAction func honkTapped(sender : AnyObject) {

}
```

第一个关键词@IBAction是来表示这是一个action连接类型，func表示这是一个方法，honkTapped就是这个方法的名字，括号内是方法的参数，括号里是参数的名字和类型，冒号后面就是这个参数的类型AnyObject。AnyObject起一个placeholder的作用，可用表示任何类型的对象。sender参数指向触发action的控件，sender参数根据控件的不同可以有很多不同的类型，像是Button，Slider，或者Segmented Control。

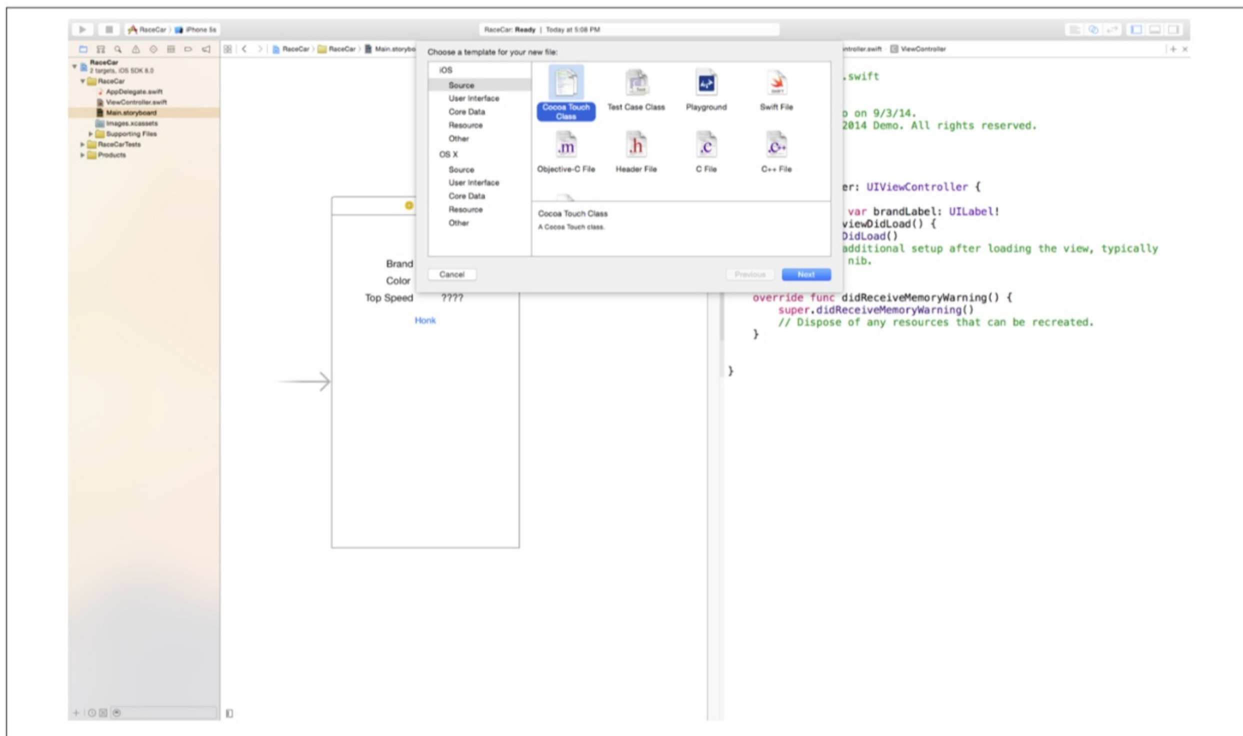
现在界面上所有需要连接的控件都已经连接到controller中了，是时候来创建你的race car了。然而，苹果并没有提供一个race car的类，没关系，你可以自己创建一个race car类。

选择File -> New -> File（见图4-30）



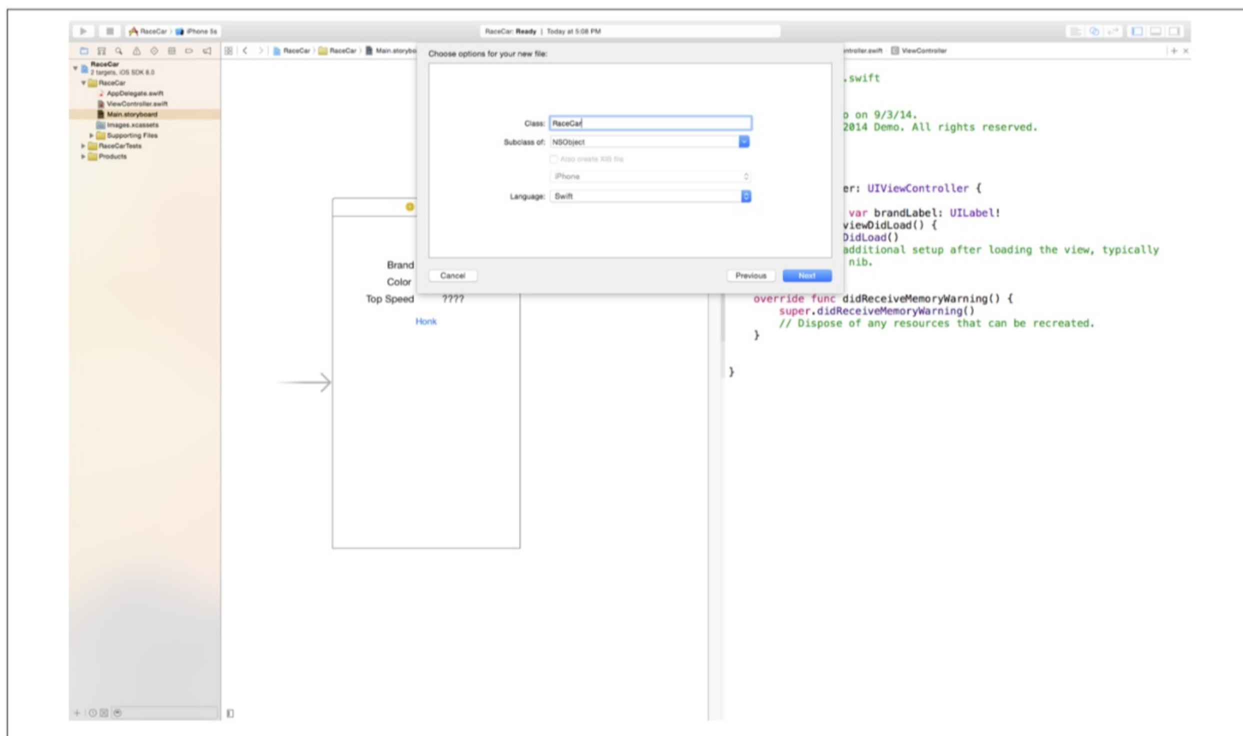
Page 124 | Chapter 4 : Diving Deeper

接着选择Cocoa Touch Class（见图4-31）。

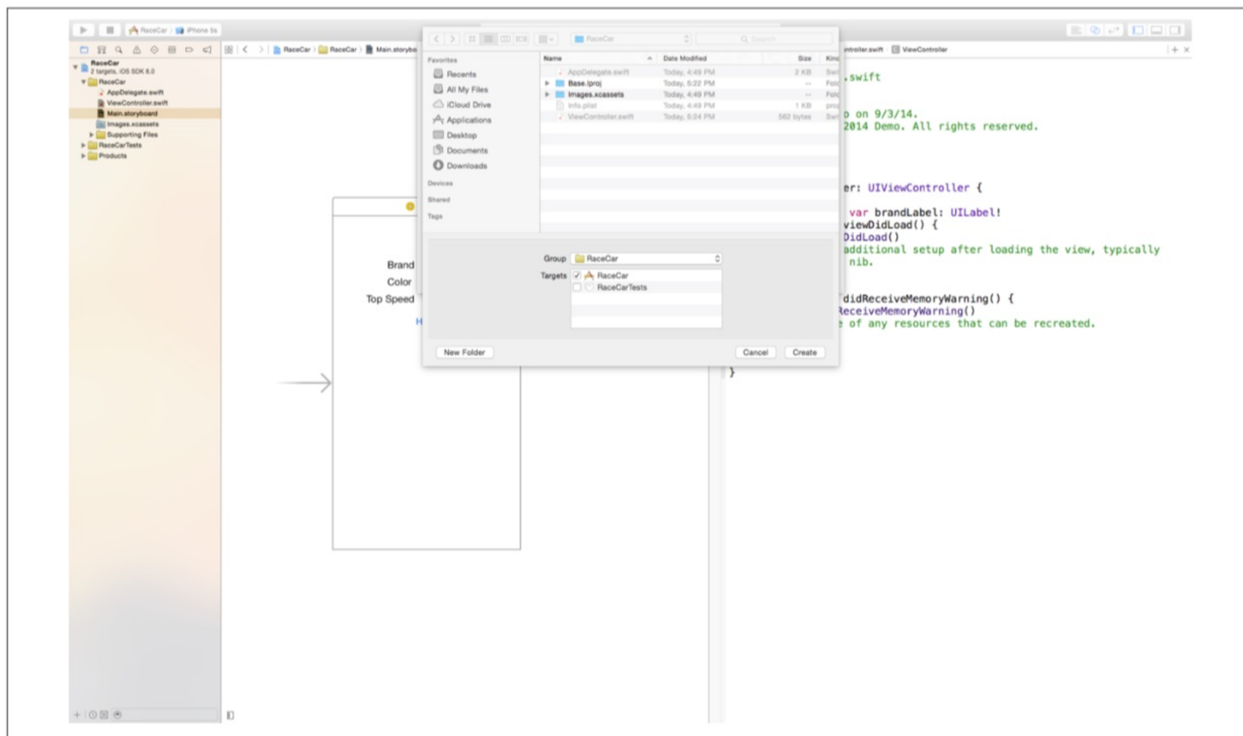


Exercise: Race Car | Page 125

在Class一栏中输入RaceCar（见图4-32），Subclass设置为NSObject，Language选择Swift，点击Next。

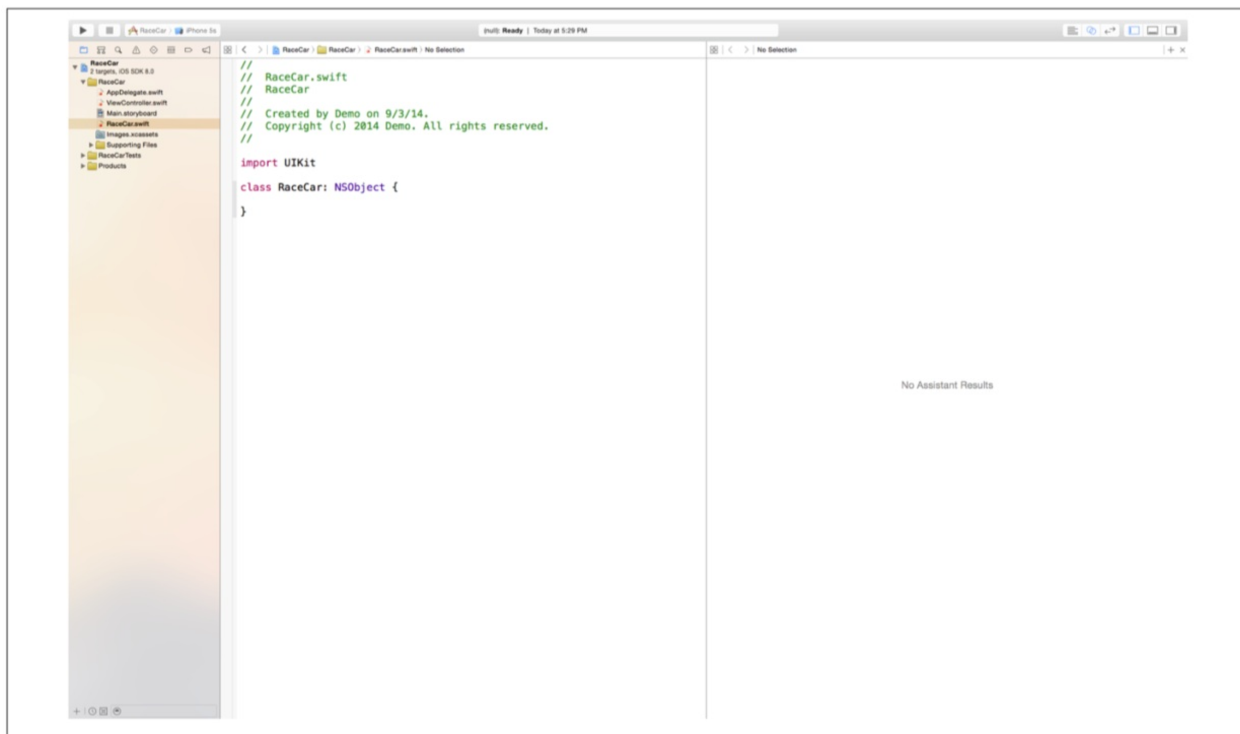


Xcode接下来会让你选择存储地点，RaceCar工程已经自动选择了，如果没有，选择RaceCar文件夹，点击Create（见图4-33）。



Page 126 | Chapter 4 : Diving Deeper

这时在Project Navigator中有了一个新文件*RaceCar.swift*（见图4-34）。



> Exercise: Race Car | Page 127

你的这个RaceCar类应该有三个属性和一个方法：brand，color，top speed以及honk方法，当RaceCar honk时，会在调试窗口打印出一行信息。开发者一般在调试窗口调试变量和事件。用户是无法看到developer log，这也是一个调试解决bug的好工具。

给你的RaceCar写下第一个属性：

```
class RaceCar: NSObject {
    var brand: String = "Ferrari"
}
```

增加属性看起来像是在创建变量，不过属性是可以让其他对象获取的。var用了创建属性，var后面跟着属性名称，接着冒号后面跟着属性的类型，等号后面是属性的值，字符串用双引号括起来。

接着增加第二个属性：

```
class RaceCar: NSObject{
    var brand: String = "Ferrari"
    var color: String = "Red"
}
```

color属性的格式和语法与brand属性相同。属性名字是color，默认值是Red，类型是String，用户双引号将字符串的内容括起来。

最后，增加第三个属性：

```
class RaceCar: NSObject {
    var brand: String = "Ferrari"
    var color: String = "Red"
    var topSpeed: Int = 200
}
```

topSpeed的属性多少有些不同了，类型是Int，因为最高时速是一个整型数字，所以默认值设置为200，属性是Int。

现在RaceCar类的属性已经都写完了，从RaceCar类中创建的车都会有brand，color和top speed这三个属性，默认值是"Ferrari"，"Red"和200。

RaceCar类还需要有一个honk行为，也就是honk方法，那么，我们把honk方法添加到RaceCar类中：

> Page 128 | Chapter 4 : Diving Deeper

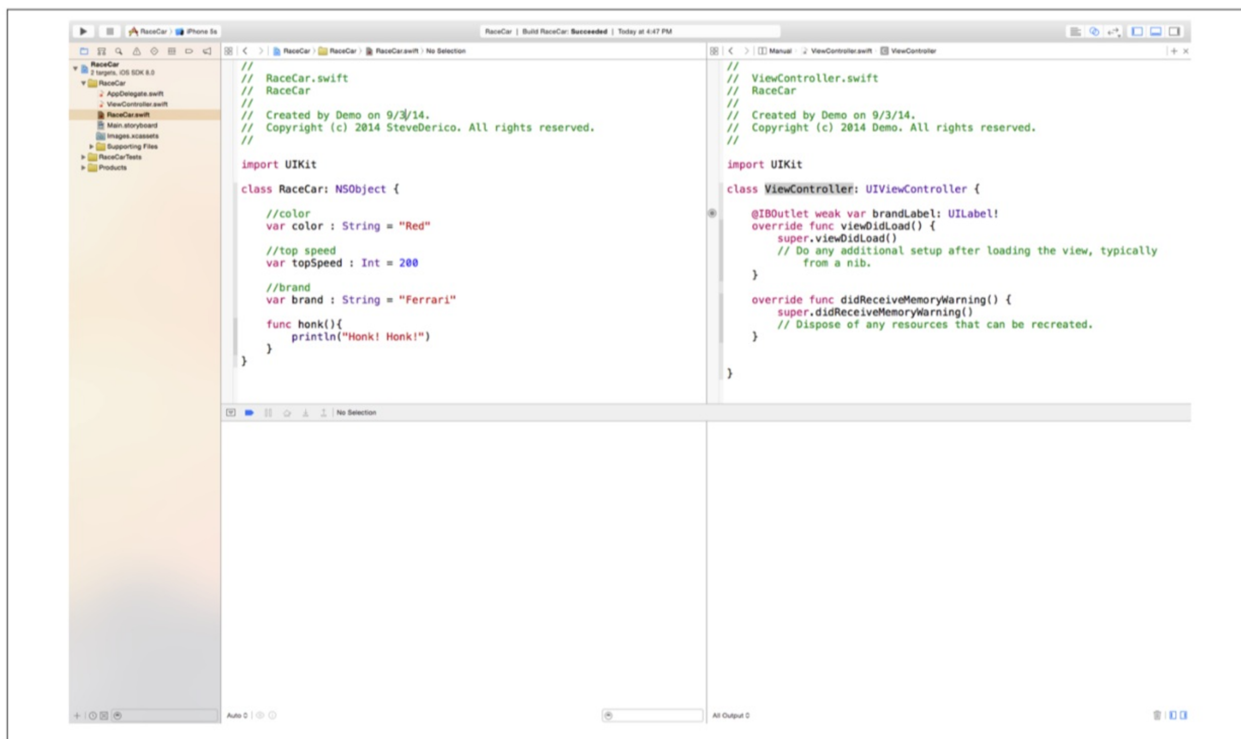
```
class RaceCar: NSObject {
    var brand: String = "Ferrari"
    var color: String = "Red"
    var topSpeed: Int = 200

    func honk() {
        print("Honk! Honk!")
    }
}
```

func关键词用来声明方法，honk是这个方法的名字，括号里是空的表示这个方法没有参数值。

在方法里面，用两个左右大括号包起来，只有一行代码，就是print这行代码，将"Honk! Honk!"打印到Debugger窗口中。

Debug Console或者Debugger（调试窗口），在写代码时候一般是隐藏的，当程序运行时，会出现（见图4-35）。打开Debugger的方法：屏幕右上角中间按钮，方块下方有一条横线的按钮，点击打开，再次点击隐藏。



现在RaceCar类已经写完了，那么，现在用RaceCar这个类来创建一辆跑车吧。快捷键Command+S保存刚刚的操作。

> Exercise: Race Car | Page 129

点击打开Project Navigator里的*ViewController.swift*文件，点击右上角Standard Editor按钮来隐藏收起Assistant Editor。

把鼠标光标放到honkTapped方法中，然后添加下面的代码：

```
@IBAction func honkTapped(sender: AnyObject) {
//Create Car
//Display Car
//Honk Car
}
```

把鼠标光标放到//Create Car这行代码后面然后敲击回车，输入下列代码：

```
var myCar = RaceCar()
```

var用了创建变量，myCar是变量名，等号表示把等号右边的值复制给等号左边的变量。最后，用RaceCar（）创建一个新的RaceCar。

RaceCar（）调用RaceCar类的初始化方法，把默认值返回给这个新的RaceCar（也就是myCar）。这初始化方法是Xcode自动生成的。从一个类中创建对象，写下这个类的名字，后面跟上一对括号。在这里的这个例子中，没有参数，所以括号内是空的。

现在这个新的RaceCar赋给了myCar变量。把鼠标光标放到//Display Car后面然后敲击回车，写下下方代码：

```
brandLabel.text = myCar.brand
```

这行代码的作用是把myCar这个变量中的brand赋值给界面上brandLabel这个控件中text属性。在这里，brandLabel上显示的文字是"Ferrari"，点这个符号可以获取Label的属性和方法，例如此处跟上了text这个属性。

myCar这个变量表示刚刚创建的新的RaceCar，后面的点符号可以获取RaceCar的属性和方法，例如此处获取了brand这个属性，这个属性返回一个string类型的值，和brandLabel的text属性类型对应一致。

在//Display后面敲击回车，输入下方代码：

```
colorLabel.text = myCar.color
topSpeedLabel.text = "\(myCar.topSpeed)"
```

你可能会注意到，colorLabel这行代码和上面的brandLabel代码非常相似。这行代码获取了RaceCar的color属性然后赋值给了colorLabel这个控件，默认值是"Red"。

> Page 130 | Chapter 4 : Diving Deeper

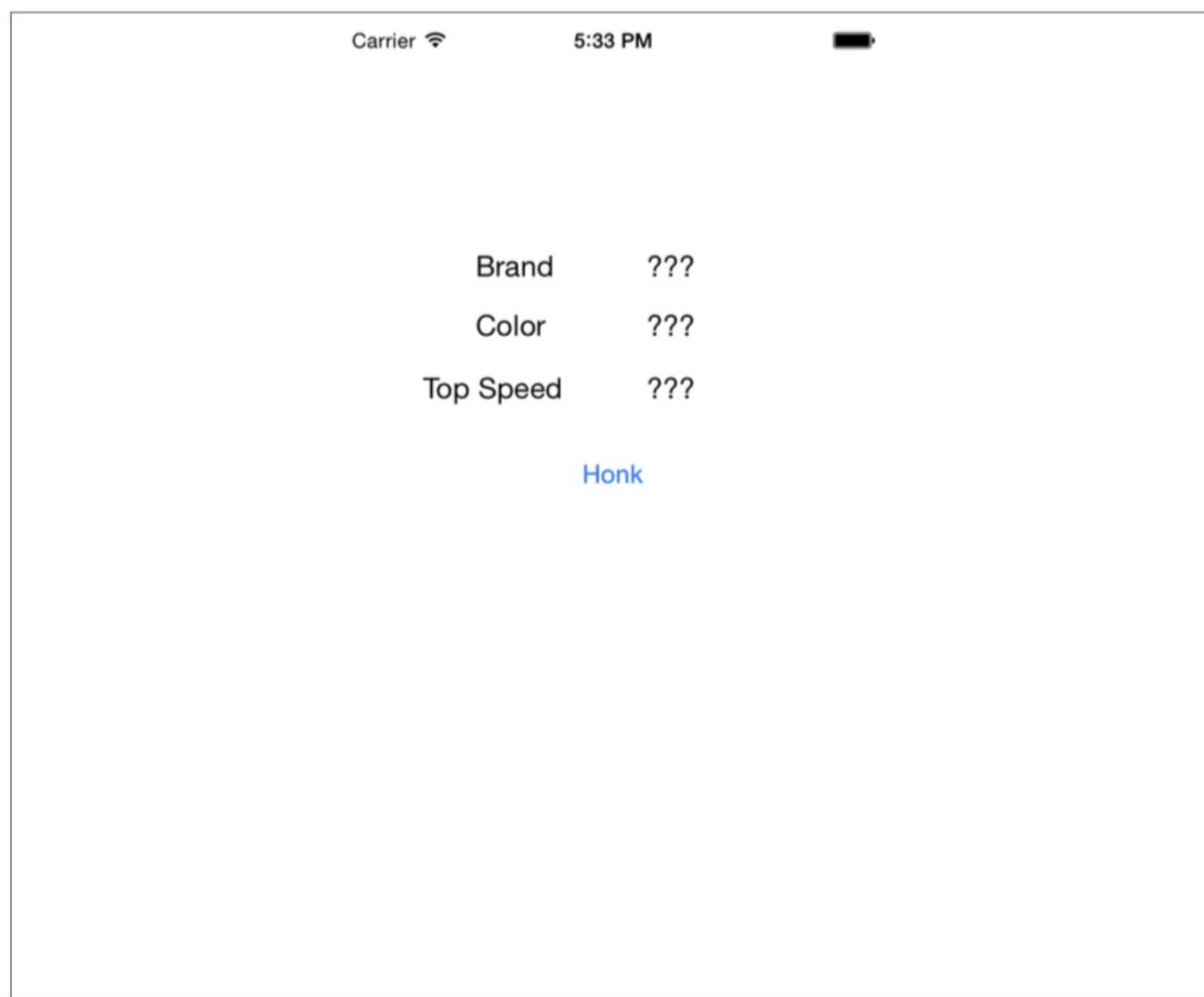
topSpeed这行代码和前面两行稍微有些不同，topSpeedLabel这个控件应该显示的RaceCar的topSpeed这个属性，Label.text需要的是string类型的值，然后RaceCar的topSpeed这个属性的值是Int类型，类型不一致，需要转换，使用()即可转换。

鼠标光标放到//Honk后面，然后敲击回车，输入下方这行代码：

```
myCar.honk()
```

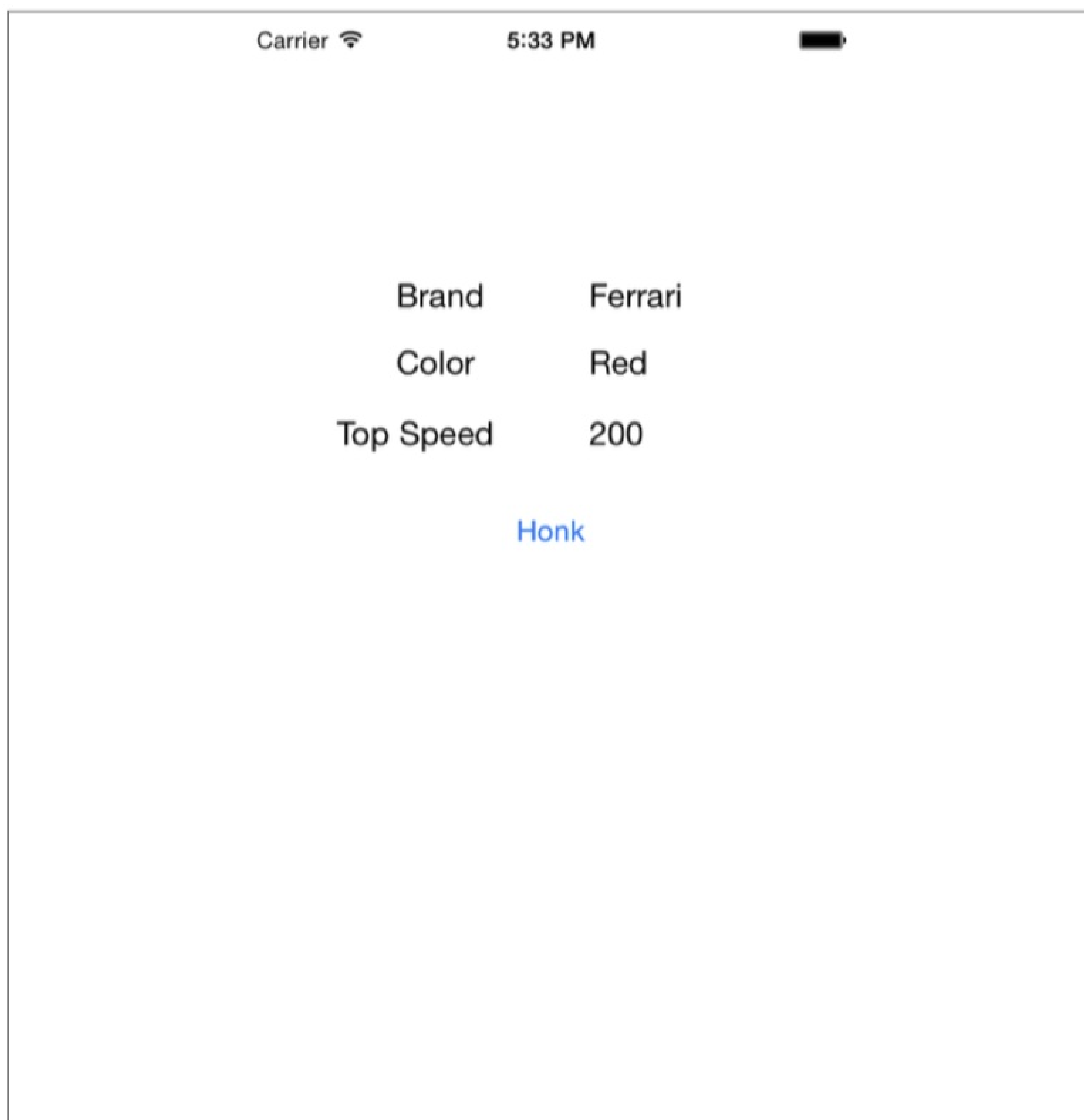
这行代码会触发RaceCar的honk方法（点符号可以获取对象的属性和方法），然后就会在Debugger中看到一行消息。括号里空的表示没有参数。

现在可以运行App看看效果了。点击左上角的Play按钮（或者快捷键Command+R），Xcode会启动模拟器（见图4-36）。



> Exercise: Race Car | Page 131

App启动后，点击Honk这个按钮（见图4-37）。



RaceCar的属性都显示在对应的Label中了，点击Honk按钮后，会调用honk（）方法，然后在Debugger中打印一段信息：“Honk! Honk!”

> Page 132 | Chapter 4 : Diving Deeper

如果App没有按照你想要的结果运行，或者程序有了错误或警告，不要太担心，学习的最佳方式就是试错，熟能生巧，到我们的网站上下载示例代码，对比一下代码，多试几次，直到搞定这个程序为止。

> Exercise: Race Car | Page 133

第五章：创建多个界面的Apps

在这一章节中，你讲学会如何在一个App里如果管理多个界面之间的跳转，还将学会如何创建一个滚动列表。 大多数的App都有多个界面和至少一个滚动列表。这章将会帮你构建知识基础，有了这些知识储备，你离着实现发布App这个目标就更近了。

视图控制器（View Controllers）

视图控制器是MVC（Modl-View-Controller）模式的逻辑部分。可以去第一章快速复习一下MVC的知识。视图控制器就是其字面的意思，这个控制器能够控制某个视图，不用过度思考。视图能够展示信息，也能够接收用户的输入。但是视图不能做决定，视图控制器来做决定。

UIViewController

苹果极力将MVC模式推荐给开发者，非常重视MVC，并且编写了自己的控制器叫做UIViewController，UIViewController是UIKit的一部分。UIKit是众多能够制作交互界面元素的类，如果你在某个类的开头是UI，那么这个类属于UIkit。UIViewController已经帮你处理了很多编程时的脏活累活和体力活，它有开箱即用的view property（视图属性），这个视图属性被连接到一个视图文件，大多数情况下，是一个storyboard文件。 UIviewController也已经为你提前写好了一些方法。一些常见的事件，诸如视图的载入和视图的消失都已经写好了，你可以把这些方法放到你自己的代码中使用。

> Page 135

总之，UIViewController提供了一些你需要的方法和属性，这样使UIViewController成为一个完美的父类，可以作为模板生成你自己的类。 例如，通常我们把UIViewController子类化，为了能够运行这个子类，在第一次运行之前，我们还需要添加一些代码进去：

```
class mySubController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()     // Do any addition setup after loading the view
    }
}
```

看一下mySubController类中的 `viewDidLoad()` 方法，override关键词告诉Xcode你要改变UIViewcontroller中默认的 `viewDidLoad()` 方法，func关键词声明了这个方法。viewDidLoad是你想从UIViewController中重写的方法的名字。 `viewDidLoad()` 方法没有参数值，最后你要加一对大括号，大括号表示方法的开始和结束位置。 不管什么时候你想重写一个方法，代码的第一行要调用super，super这个关键词是用来表示目前这个类的父类。在这个例子中，父类

就是UIViewController。调用super是非常重要的一步，如果不调用super，在你的代码被执行之前，`viewDidLoad()` 方法是不会做什么事情的，就好像你要建房子，却没有先打地基。所以用 `super.viewDidLoad()` 来打地基，打好地基后，你就可以增加你自己的代码：

```
class mySubController: UIViewController {
    override func viewDidLoad() {
          super.viewDidLoad()
          // Do any addition setup after loading the view
          println("The View Has Loaded")
    }
}
```

println方法调用后将会在Debugger中打印出一段信息"The View Has Loaded"。在编程时，记着给每一个视图控制器都配置一个视图和控制器，这是个好习惯。

UINavigationController

直到现在，你编写的App都只有一个界面。然而，现在iOS平台上知名的App都有多个界面，能够从左过渡到右边。例如，iPhone上的邮件App，当你点击一个邮件时，会滑动到一个新的界面。

> Page 136 | Chapter 5 : Building Multiscreen Apps

邮件App也有返回按钮。这种从左到右的过渡效果是navigation controller创建的。

一个UINavigationController可以在数组中支持多个UIViewController。记着，数组是集合类型，能够按照某个顺序存储多个元素。导航控制器（navigation controller）的使用的排序机制叫做堆栈（stack），按照先进后出的原则排列元素。stack有些像是登机，如果你第一个登上飞机，走到飞机后面坐在靠窗的位置，那么在下飞机时，你是最后一个离开飞机的。这就是先进后出原则在工作时的一个范例。 假设你第一个登上飞机，走到飞机后面坐在靠窗的位置，接着一个男性乘客登机坐在了你旁边的位置，接着一位女性乘客登机，坐在了靠近走廊的位置。那么当离开飞机时，先是这位女乘客立刻飞机，接着是男乘客，最后才是你。堆栈按照相反的顺序加载和卸载，加载的时候是由下到上，卸载的时候是由上到下。 为了给navigation controller增加view controller，你首页要把view controller push推到navigation controller的上面，push就相当于第一个人正在登机。The item goes as far toward the bottom as possible, but does not pass the itemsinserted before it. （这句话不会翻译，静等高手）。堆栈中最上面的view controller将被展示给用户。为了能给用户展示另外一个界面，就把另外一个界面push到当前的界面。 为了去掉当前的界面，我们使用navigation controller中的pop功能。（push和pop在ios9中已经被弃用了）pop是指从堆栈中移除最上面的元素。在这个例子中，pop移除的是当前展示的界面，然后回到之前的界面。pop相当于头等舱的乘客，最后一个登机，第一个离开飞机。 当一个view在navigation controller中展示时，会在这个view的顶部增加一个navigation bar。这个navigation bar有三个主要的部分：左边的UIBarButtonItem，中间的titleView，然后是右边的UIBarButtonItem，UIBarButtonItem就像是一个典型的button，但是它是在UINavigationBar中，UINavigationBar这个类用来生成

navigation controller中的navigation bar，当另外一个界面展示在navigation controller中时，左边的UIBarButtonItem就会自动变成一个返回按钮。titleView展示UIViewController的title属性值，这里关键记着，title属性值是由view controller定义的，但是是在navigation controller中显示。在view controller中设置title属性非常简单：

```
title = " Countries"
```

每一个view controller都有title属性，因为UIViewController定义了这个属性。返回按钮也会显示要返回的界面的名字。

> View Controller | Page 137

最后，右边的UIBarButtonItem是可选的，右边这个位置可以放置一个主要或者次要的行为。例如设置按钮可以放在App的右上角。

Table View

滚动视图是iOS Apps中最常见的用户界面。例如，iPhone的设置，里面就有一个目录滚动视图，当用户点击某个选项时，会跳到新的界面显示更详细的信息。这个效果是结合了navigation controller和table views来实现的。navigation controller控制着目前显示哪个界面，而table views则显示滚动视图内容。

table view有几个关键部分组成，滚动视图中的每一行叫做cell，cell是用了展示table view中每行的内容。table view可以有很多个cell，多个cell组成section（组）。sections是用来把行区分成不同的部分。在iPhone设置，就是用不同的section把目录分开，像是通知中心，控制中心，个人隐私。每个table view都有header和footer，header是在cell上面，footer在cell下面。

最后，一个table view有两种style（风格）。默认风格是Plain，一个紧挨着一个列出每一行，另外一个风格Grouped，是把一组一组在一起，不同的组之间用空隙间隔。

Delegation

很多的体育网站或者是App都会提供一个提醒的服务，当你喜欢的球队得分时，就会往你的手机或者邮箱中发送提醒的消息，比如不管什么时候旧金山巨人队赢了全垒打，你都会收到震动提醒。 类似的概念也应用在了控制器的提醒功能上，App内部发生某个事件时，就会发出提醒。为某个事件订阅或者接收提醒的过程叫做delegation（委托）。委托一个delegate接收所有的通知。然而，delegate不只是只接受通知，大多数情况下，delegate也必须回应这些通知。 例如，你要管理一个公司，你告诉保安处监视休息室，不管什么时候休息室的人数超过3个，就让保安敲你的门。敲门就是一个通知，在每次收到通知后，你可以选择如何处理这些通知。

当一个table view第一次创建时，需要回答一系列的问题后才能使用。table view让delegate来接收问题并提供答案。

> Page 138 | Chapter 5 : Building Multiscreen Apps

例如，table view会问delegate要创建多少行，delegate必须用个具体的数字或者整型变量来回答这个问题。 实际上可以为每个需要知道的信息重写方法就能回答一系列的问题。例如，问要有多少行，你需要重新 `numberOfRowsInSection` 方法：

```
override func tableView ( tableView: UITableView, numberOfRowsInSection section: Int ) ->
 //Return the number of rows in the section
return 10
}
```

如何看不懂方法名字，也没有关系，关注方法里面的代码就可以了。在这个例子中， `numberOfRowsInSection` 方法返回了一个整型，返回的是10，这样table view就会有10行。这就是众多必须回答的问题之一，回答完这些问题才能创建一个table view。 所有的问题都打包在一个协议（protocal）里。协议（protocal）以一种特定的方式做事。例如，机场的协议就是要检查你的行李，通过安检，登机。这每一个步骤，你都要提供信息，如目的地是哪里。关键是你要完全同意协议中要求的事项。在编程时，这就叫做conforming（遵从协议），表示你同意回应协议中所有需要回应的方法。

UITableViewController

UIViewController 同样的，UITableViewController也是已经为你创建table view做了大部分的脏活累活体力活。UITableViewController会自动创建一个table view，然后设置 `tableView` 属性，同时也需要委托自己获取所有需要的delegate方法。

UITableViewDataSource

UITableView的delegate协议有三个必须要写的方法，叫做UITableViewDataSource。这个协议包括组的数量，美组中行的数量，以及cell如何展现。

> Delegation | Page 139

第一个方法是 `numberOfSectionsInTableView(_:)` ，这个方法需要一个整型值，来作为table view中section组的数量。重写这个方法非常简单：

```
override func numberOfSectionsInTableView(tableView: UITableView) -> Int {
    //#warning Potentially incomplete method implementation.
    //return the number of sections.
    return 0
}
```

注意到return那行目前是零，这意味着这个table view中没有组。苹果公司增加了一个警告注释，说如果组的个数是零，那么就不会显示行，组包含行cell，没有了组section，行cell也就不会被显示出来：

```
override func numberOfSectionsInTableView(tableView: UITableView) -> Int {
     //return the number of sections.
    return 1
}
```

第二个方法是 `tableView(_:numberOfRowsInSection:)` ，这个方法需要一个整型，决定了某个组里具体有多少行：

```
override func tableView(tableView: UITableView,numberOfRowsInSection section: Int) -> Int
     //#warning Incomplete method implementation
    // Return the number of rows in the section
    return 0
}
```

注意到return后面是零，这意味着每个组里都没有行，所以苹果公司增加了一个警告注释，让你重写这个方法：

```
override func tableView(tableView: UITableView,numberOfRowsInSection section: Int) -> Int
     //#warning Incomplete method implementation
    // Return the number of rows in the section
    return 5
}
```

最后第三个方法是 `tableView(:cellForRowAtIndexPath:)` ，这个是和行cell有关。这个方法里有个参数值叫 `indexPath` ，是一个NSIndexPath。NSIndexPath有两个属性。

> Page 140 | Chapter 5 : Building Multiscreen Apps

section组属性的索引是当前组，cell行属性的索引是当前行。记着索引是从零开始计数的：

- 第一组第一行的索引NSIndexPath是0,0。
- 第一组第四行的索引NSIndexPath是0,3。
- 第三组第一行的索引NSIndexPath是2,0。

可以用点语法调用section和row属性：

```
var currentRow = indexPath.row
var currentSection = indexPath.section
```

`tableView(:cellForRowAtIndexPath:)` 这个方法一开始会让有些你灰心丧气，不过不用担心，因为过一会就会明白了。先值关注方法里面的代码，不要害怕犯错误：

```
override func tableView(tableView: UITableView,cellForRowAtIndexPath indexPath: NSIndexPa
    let cell =tableView.dequeueReusableCellWithIdentifier("Cell", forIndexPath: indexPath

    return cell
}
```

Cell reuse（cell 重复使用）

第一行先创建了一个名为cell的常量。cell等同于一个table view的一个离队行（dequeued cells）。一个离队的cell是一个不再使用的cell。就和真实的世界一样，循环利用资源非常重要。当table view有很多行，受屏幕限制能一次显示出来的行数是有限的，这些还没有显示出来的行实际上还没有创建出来。当这行滑出屏幕后，就进入了回收箱。在新的行创建之前，回收箱会检查一下有没有可回收利用的行。正是因为cell的可重复使用，在使用之前先删除之前的信息，就变得格外重要了。

通过 `tableView` 变量来访问table view， `dequeueReusableCellWithIdentifier` 方法使用回收来的cell或者创建新的cell。 `dequeueReusableCellWithIdentifier` 方法有两个参数值： `identifier` ，String类型，确认具体哪一行， `IndexPath` 是NSIndexPath类型，标识具体位置。

cellForRowAtIndexPath方法用override开头，因为这个方法是继承自UITableViewController。有了override，可以重写cellForRowAtIndexPath方法了。

> Delegation | Page 141

func关键词表示这是一个方法。第一个tableView是这个方法名字的一部分。括号中有这个方法需要的参数，第一个参数名字是tableView，是一个UITableView。cellForRowAtIndexPath是这个方法的名字，接下来的参数名字是indexPath，是一个NSIndexPath。最后，这个方法需要一个UITableViewCell作为返回值。

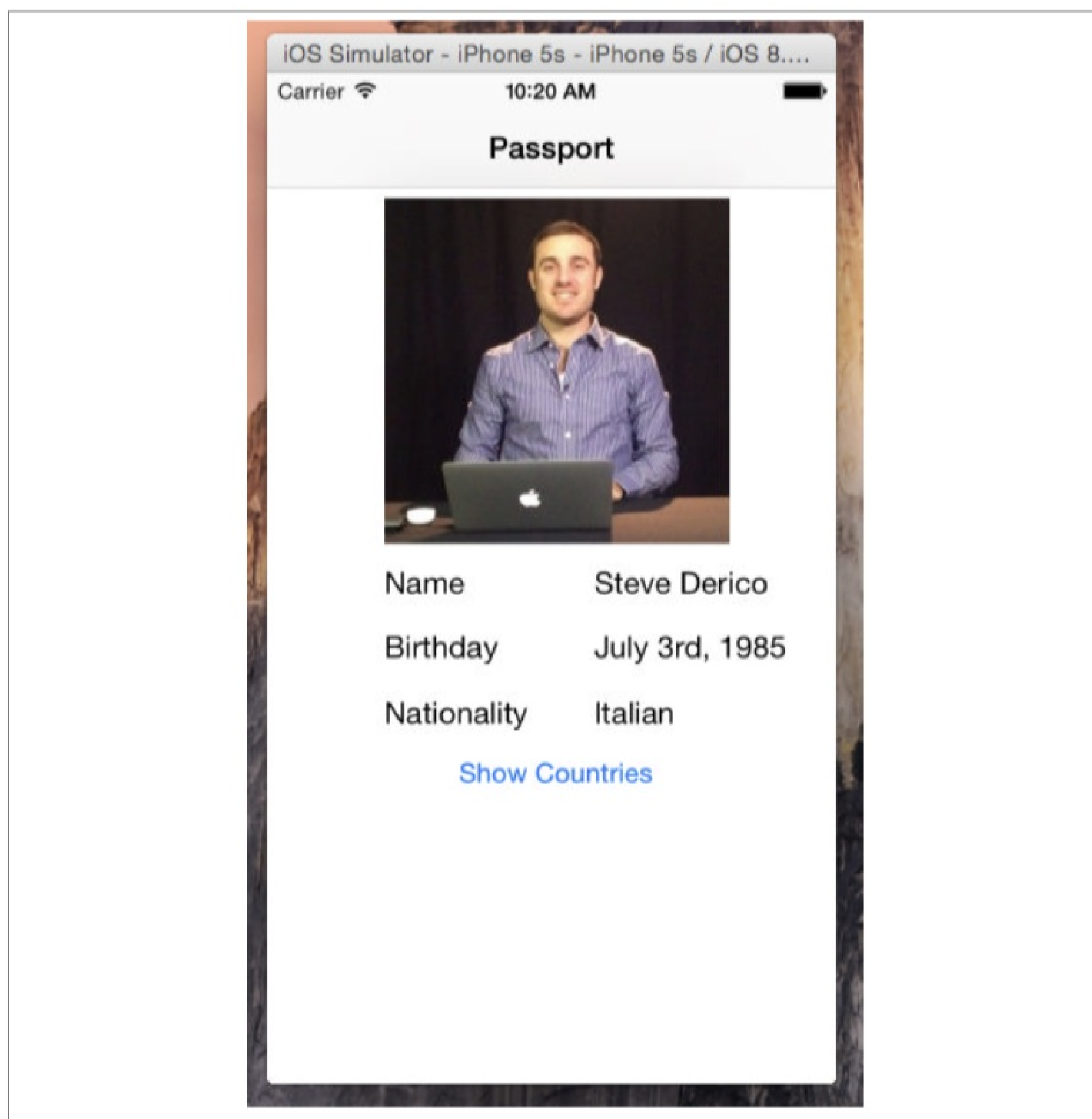
一旦你成功创建cell，接下来就是为cell设置属性来展示你的数据。 UITableViewCell有5类主要属性： 1.textLabel，属于UILabel，展示主要信息； 2.detailTextLabel，属于UILabel，展示副标题，这个label不总是出现的； 3.imageView，属于UIImageView，在cell的左边显示图片； 4.accessoryView，显示一个大于号； 5.contentView，空白cell，可自定义。

UITableViewCell有4种风格（style），上面的5个属性在不同的风格下有可能被隐藏或者位置变化： 1.Default 左对齐的text label，可选的imageView，没有detailTextLabel 2.Value1 左对齐的黑色字体的text label，一个更小的蓝色字体右对齐 3.Value2 左侧有个右对齐的蓝色字体的text label，右侧有个左对齐的黑色字体的text label 4.Subtitle 左侧左对齐的黑色字体的text label，接着是一个字体更小的左对齐灰色text label

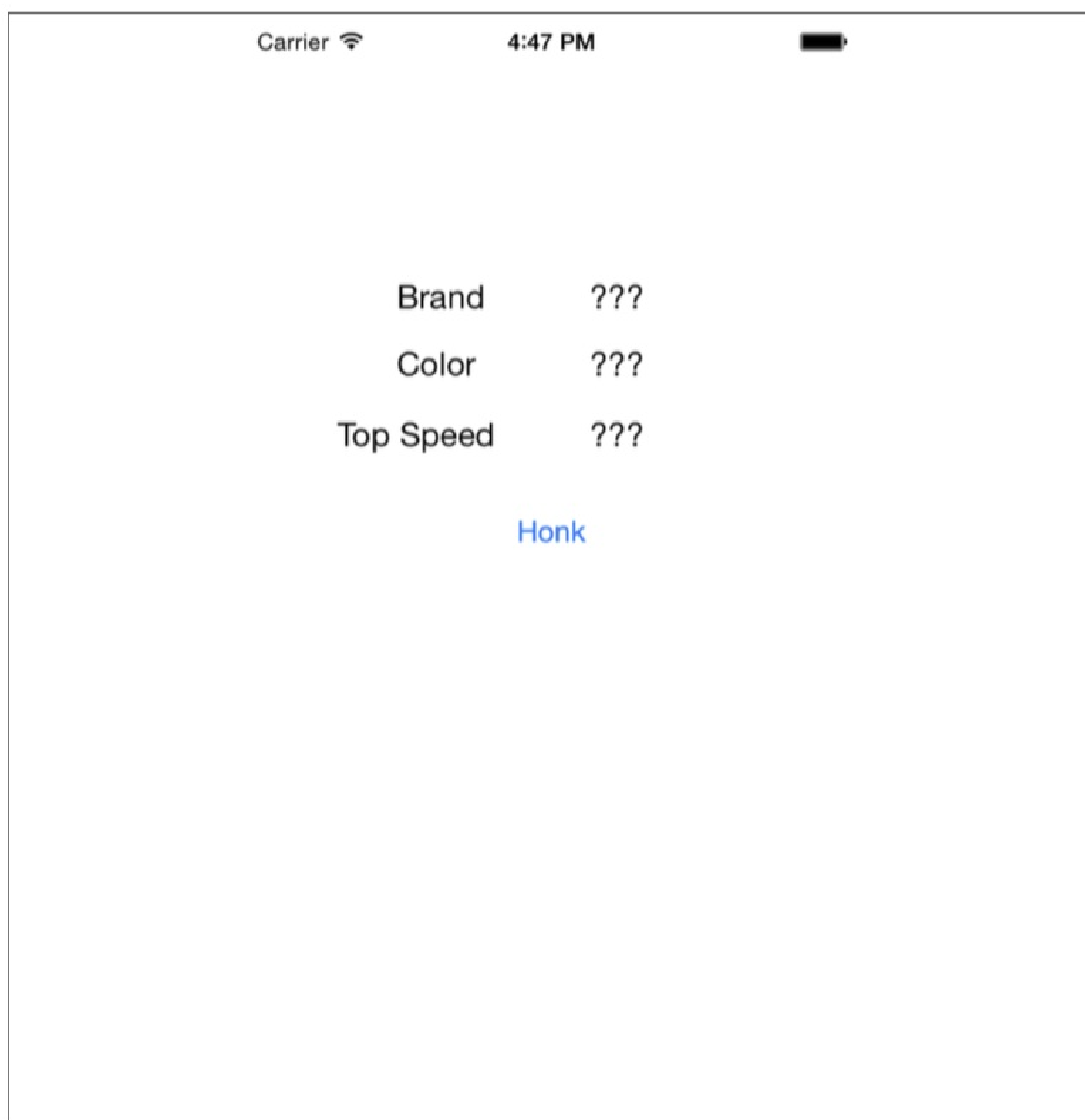
在这一章节中，你已经学会了在一个APP里管理多个界面，还学会了如何创建一个滚动视图，你的知识库正在充盈，现在是时候把刚刚学会的东西应用一下了，开始我们的Passport练习吧。

> Page 142 | Chapter 5 : Building Multiscreen Apps

第五章练习 Passport - 编写一个护照App吧

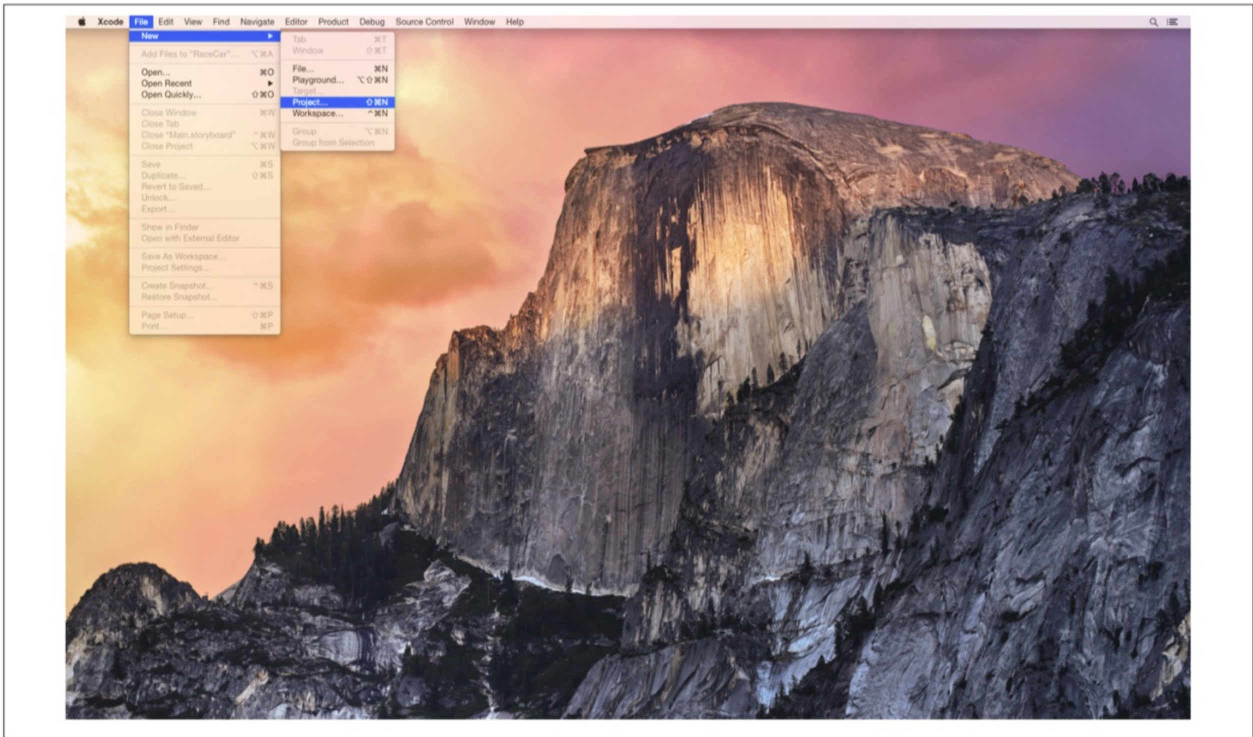


在这章练习叫做Passport，Passport是一个非常简单的应用，展示个人的名字、生日、国籍和照片（图5-1）。Passport还提供各种国家供你选择，显示曾经去过哪些国家。这个练习将会在下一章中继续使用。

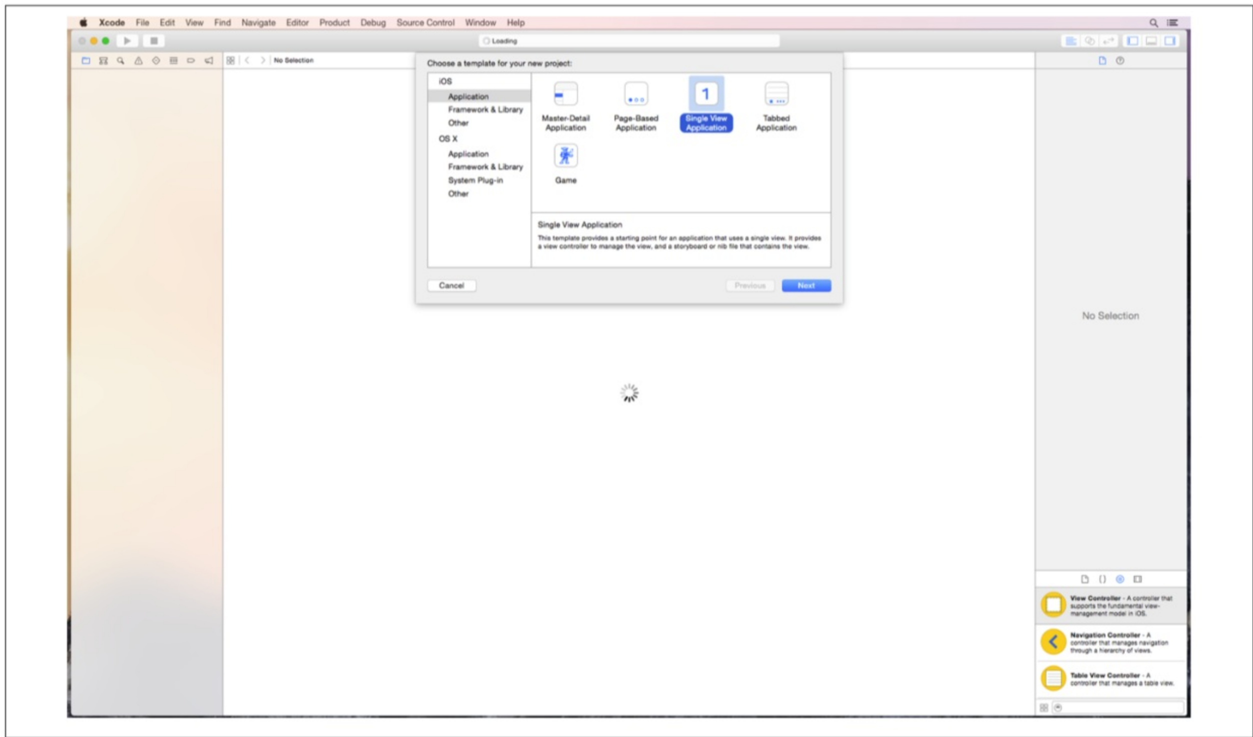


Exercise: Passport | Page 143

打开Xcode吧，点击顶部菜单栏的File -> New -> Project（见图5-2）。

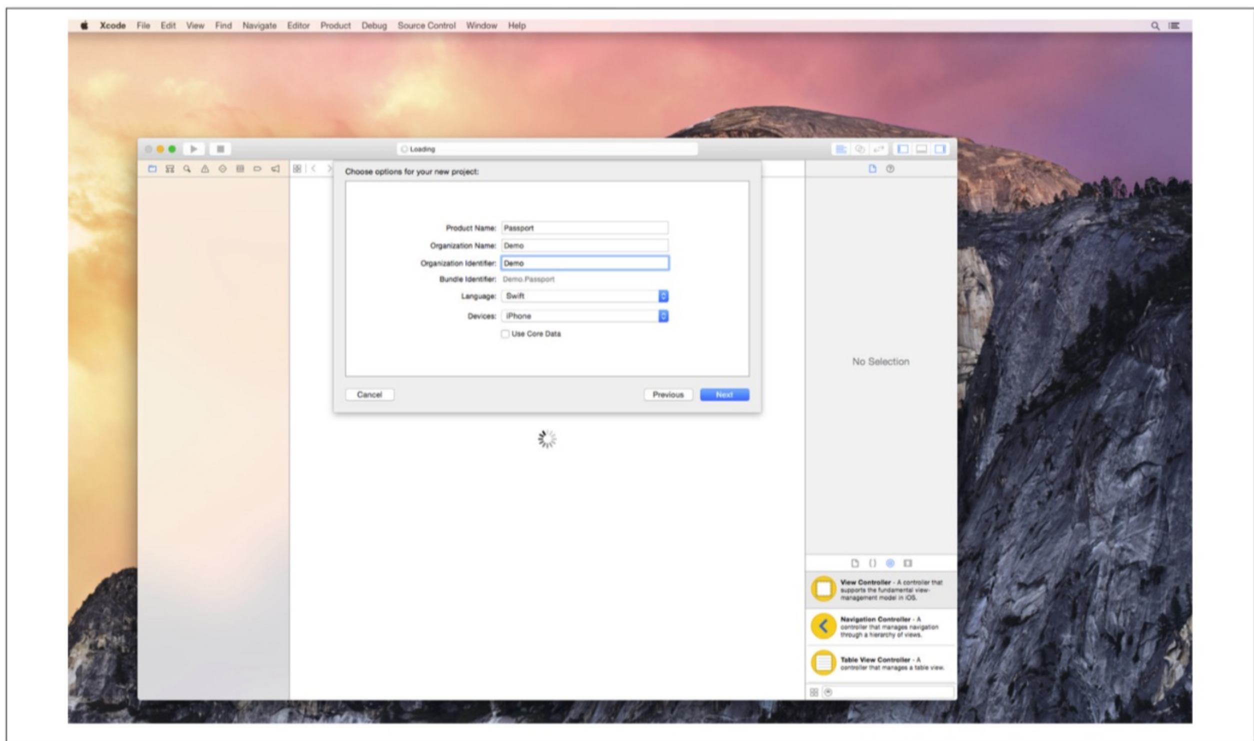


从模板中选择Single View Application，点击Next（见图5-3）。

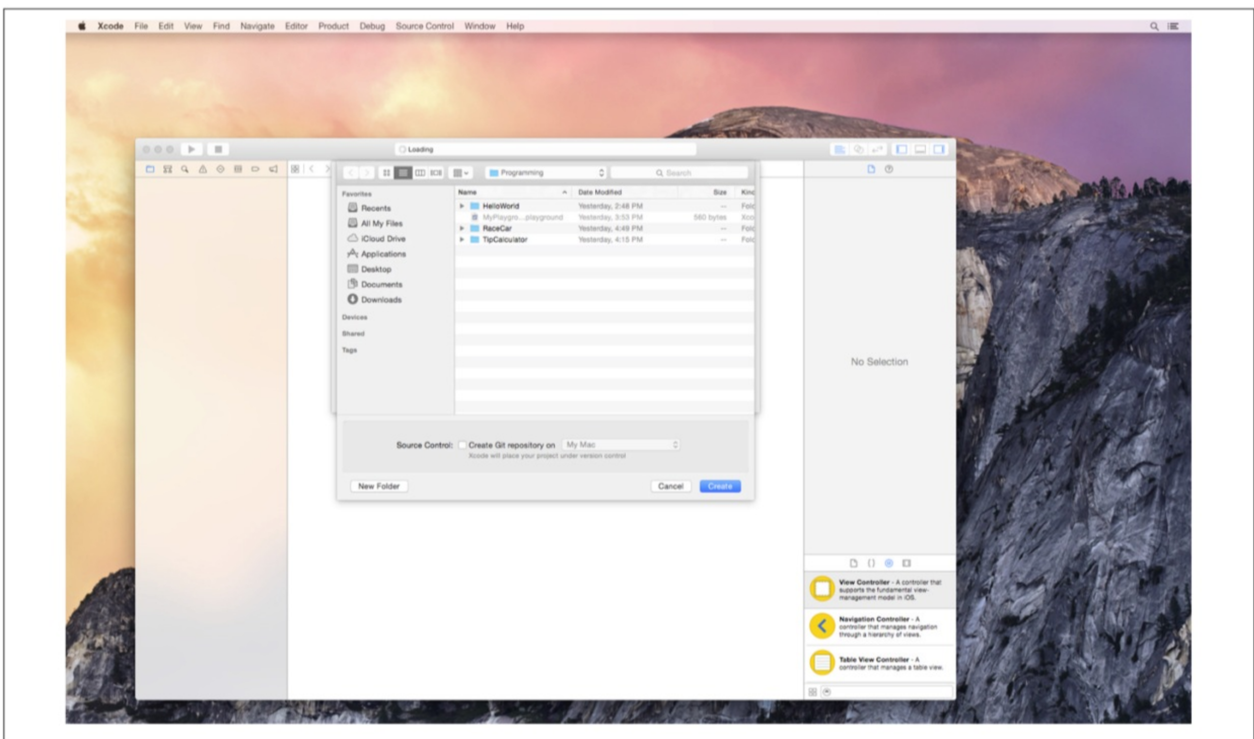


> Page 144 | Chapter 5 : Building Multiscreen Apps

在Product Name一栏输入Passport，Language选择Swift，Devices选择iPhone（见图5-4），点击Next。

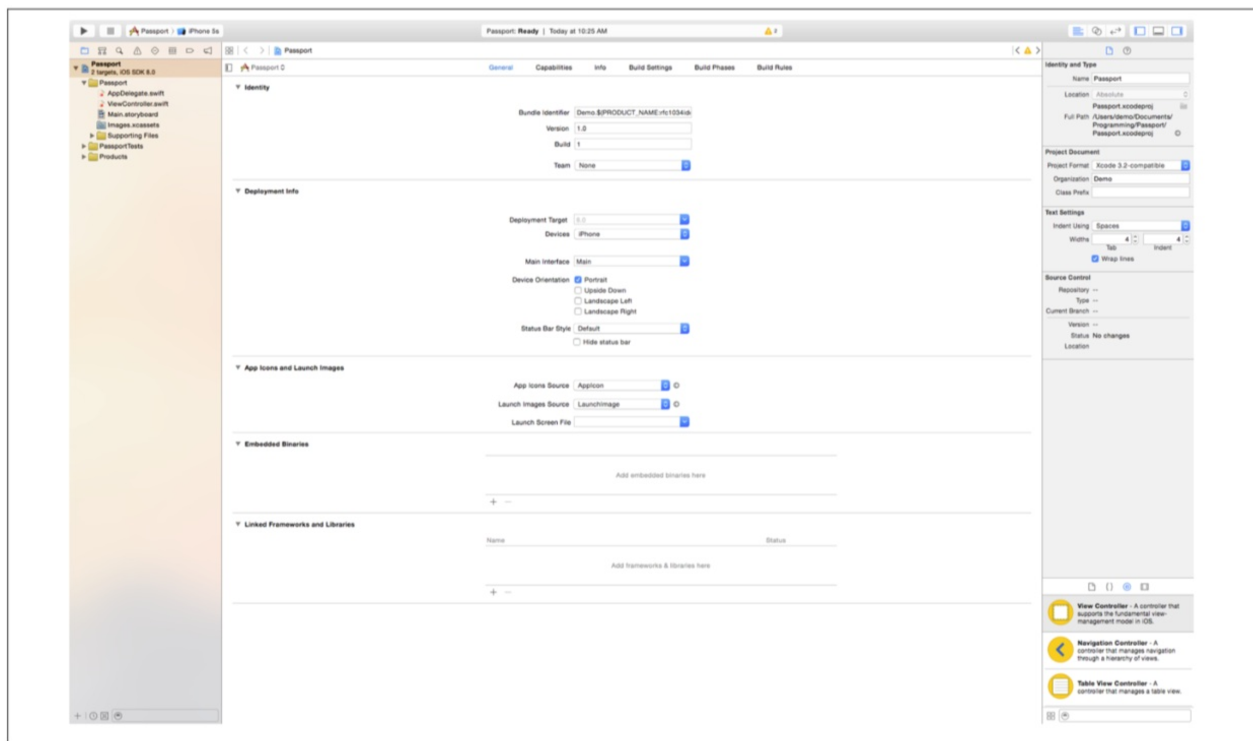


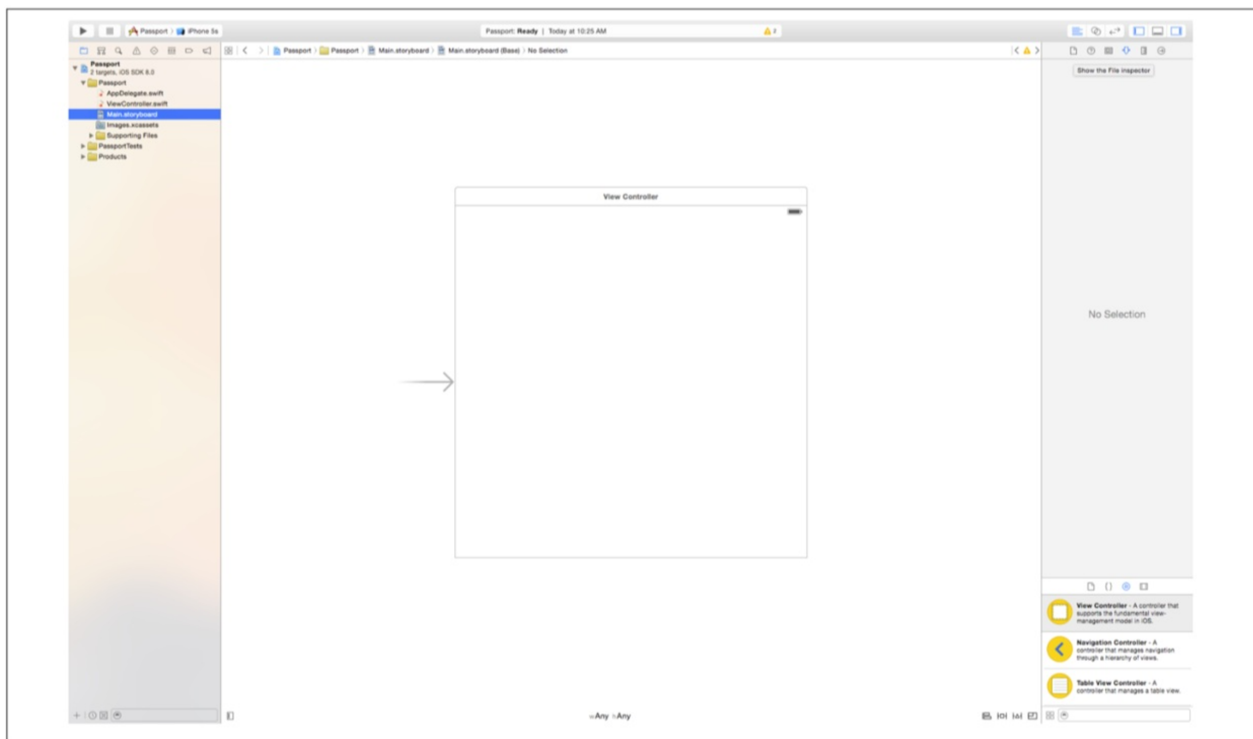
把工程保存到某个文件夹中（见图5-5）。



Exercise: Passport | Page 145

出现了工程的详细信息界面（见图5-6）。不勾选Landscape Left和Landscape Right这个两个方向，然后打开Main.storyboard（见图5-7）。



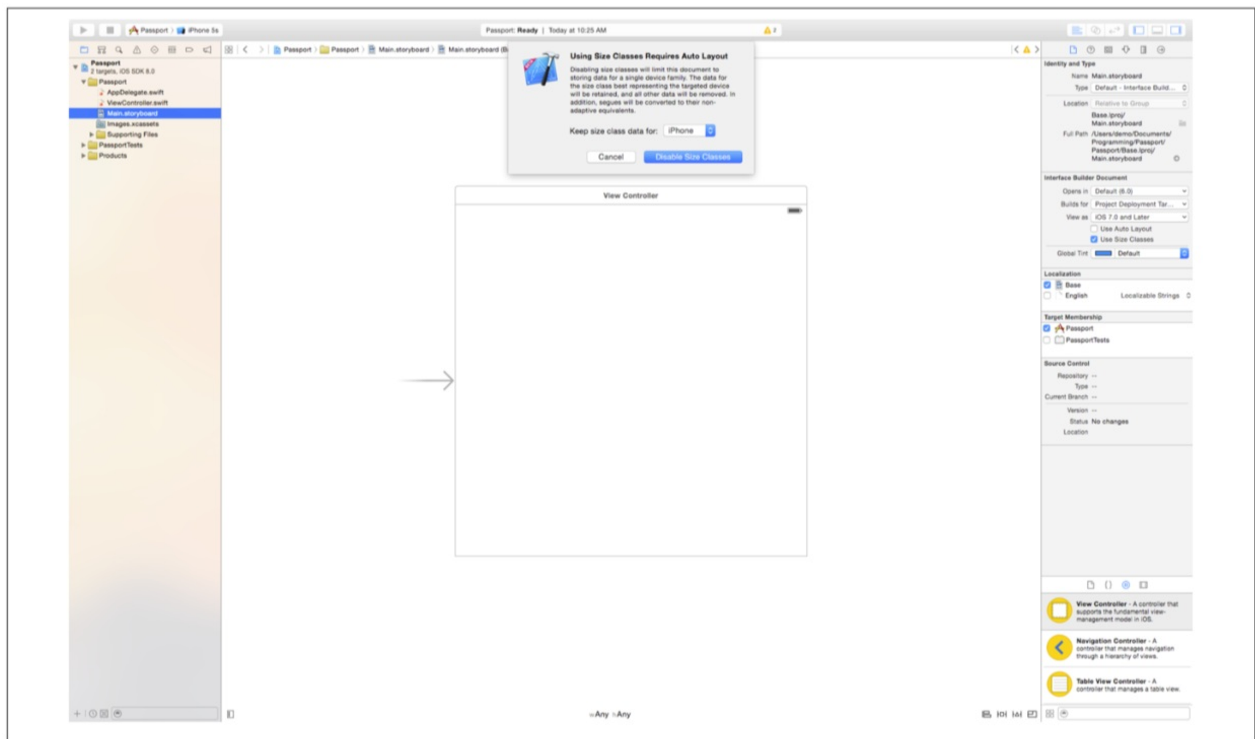


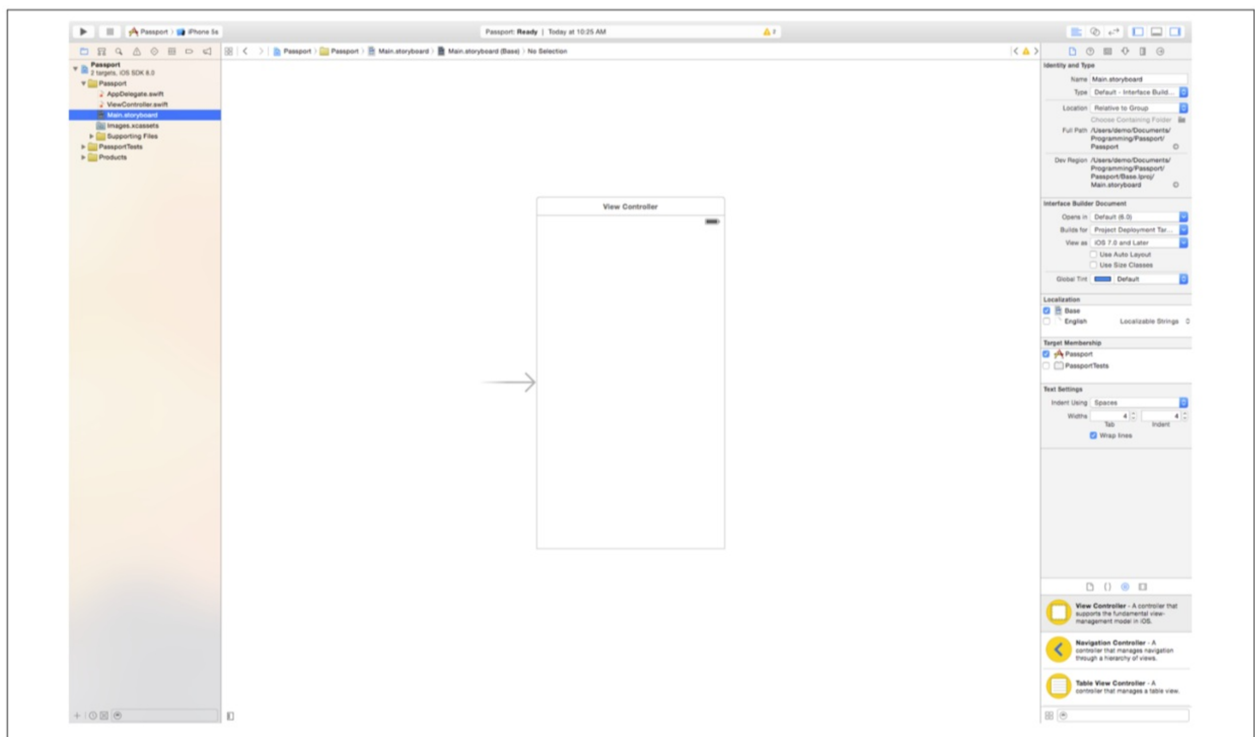
Page 146 | Chapter 5 : Building Multiscreen Apps

点击Inspector上工具栏中第一个图标File Inspector，看起来像是一张纸折了一个角。

鼠标移动到到中间部分，不勾选Use Auto Layout选项。这时会出现一个对话框，选择iPhone。然后不勾选Disable Size Classes。这时，Storyboard中的界面形状会改变（见图5-8和图5-9）。

Exercise: Passport | Page 147



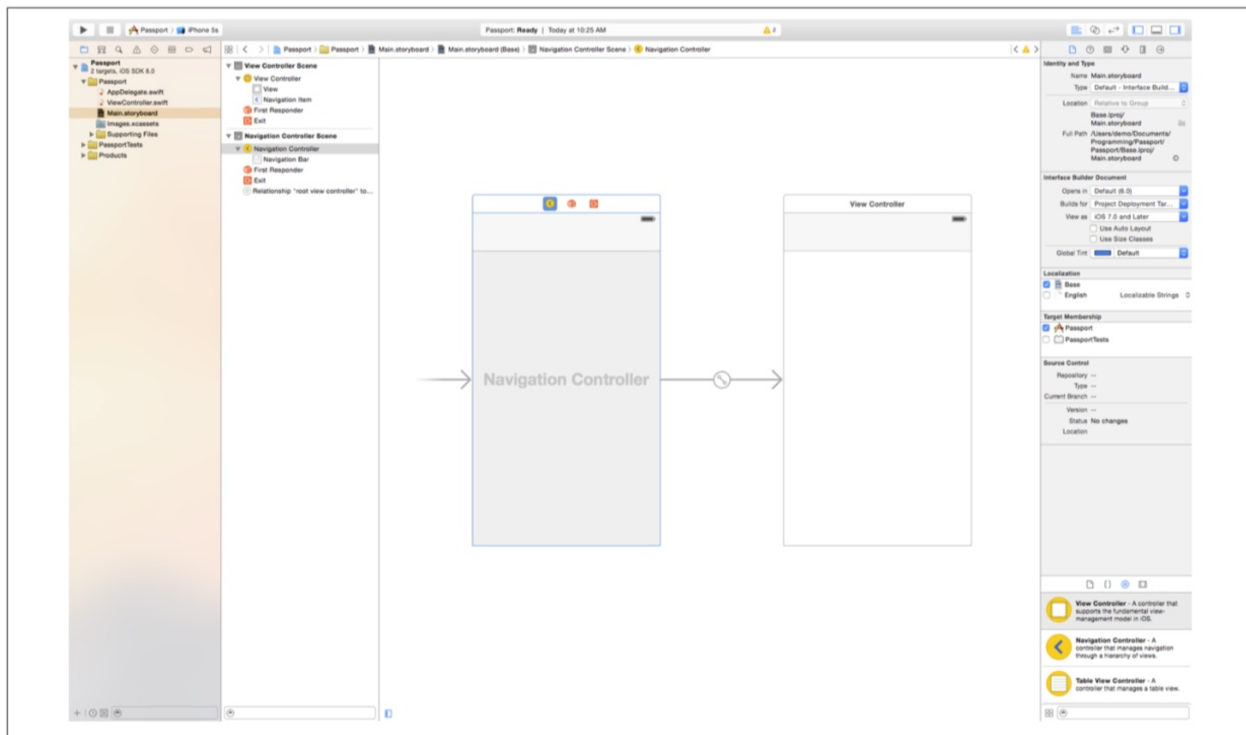


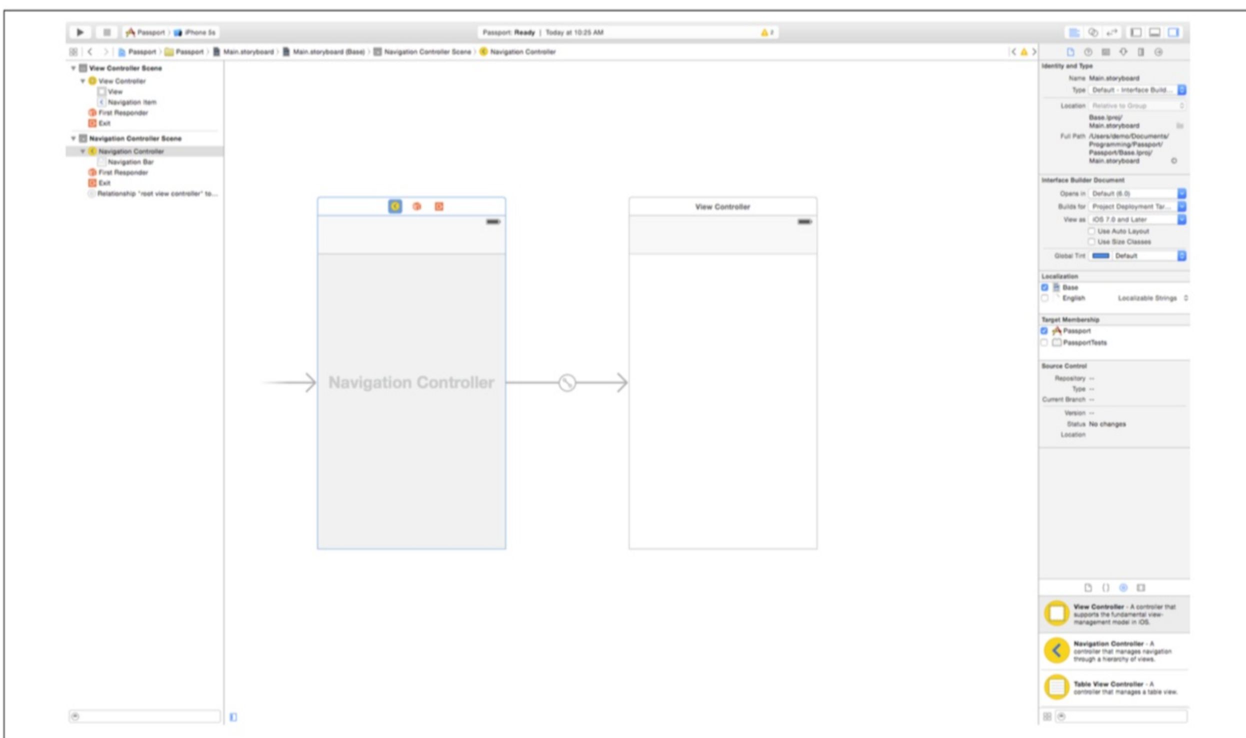
选中这个界面，然后点击顶部菜单栏的Editor -> Embed In -> Navigation Controller。

> Page 148 | Chapter 5 : Building Multiscreen Apps

一个新的scene会增加到Storyboard中，Navigation Controller Scene（见图5-10）。一个scene表示App一屏或者一个界面。Navigation Controller Scene和之前的View Controller Scene是连接在一起的，这连接说明View Controller Scene是Navigation Controller Scene里

第一个出现视图，点击Storyboard Editor左下角的盒子按钮打开Document Outline，Document Outline显示了storyboard文件中所有的控件以及控件所处的层次等级。接着把Project Navigator隐藏起来（见图5-11）。

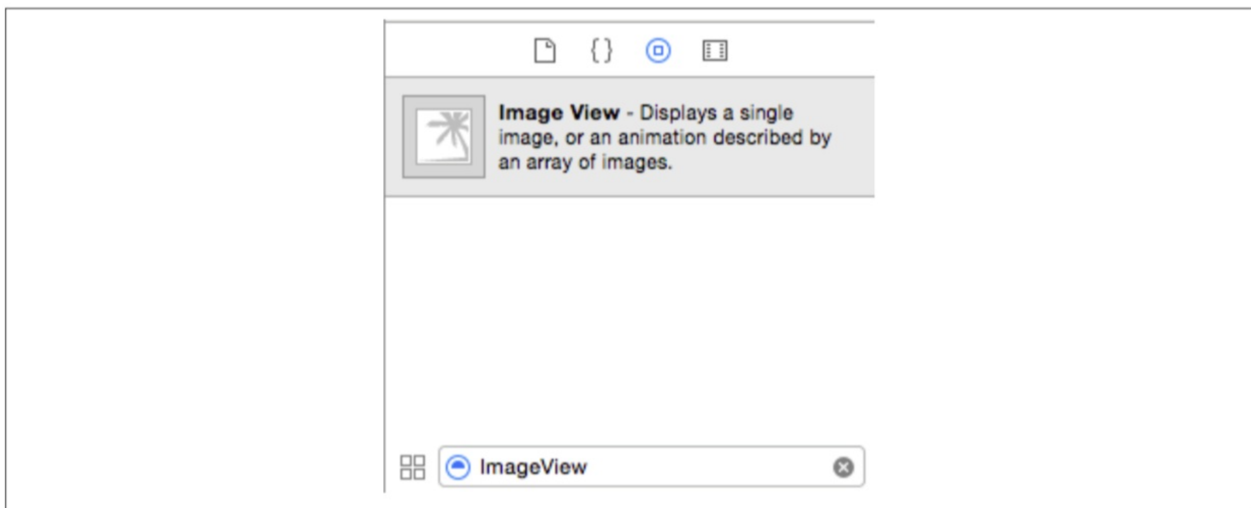




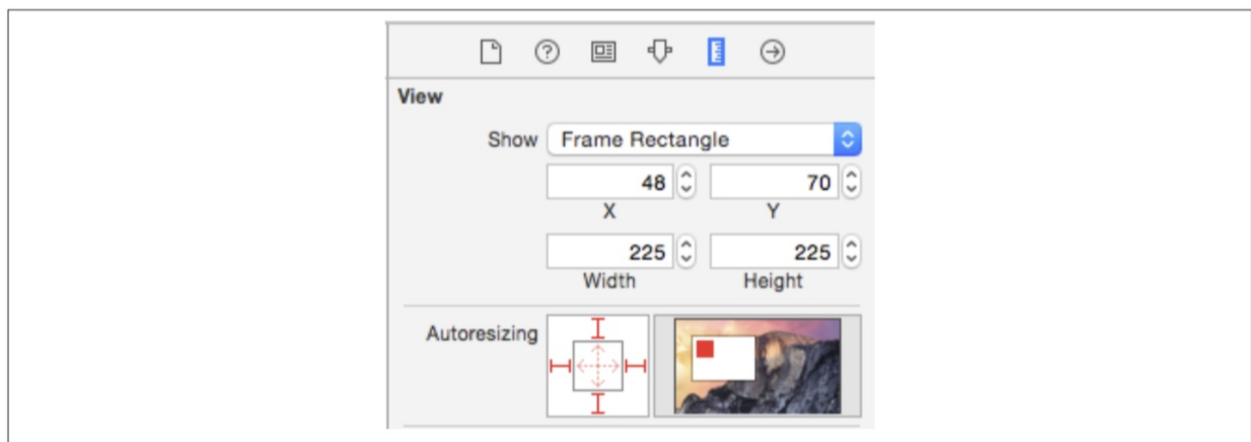
Exercise: Passport | Page 149

View Controller Scene的顶部有个灰色的方块，这就是navigation bar。双击，然后输入Passport，敲击回车。

在Inspector中间打开Object Library，在搜索框中输入Image View（见图5-12）。

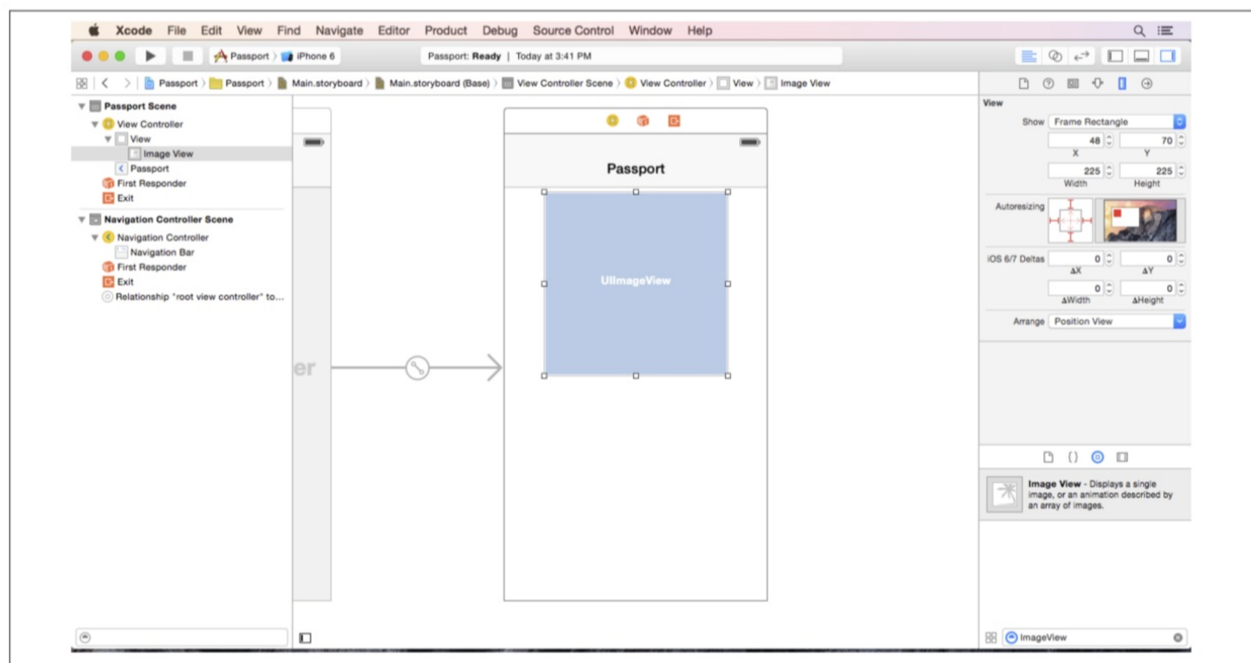


把Image View拖到View Controller界面的中间，然后打开Size Inspector（见图5-13）。



Page 150 | Chapter 5 : Building Multiscreen Apps

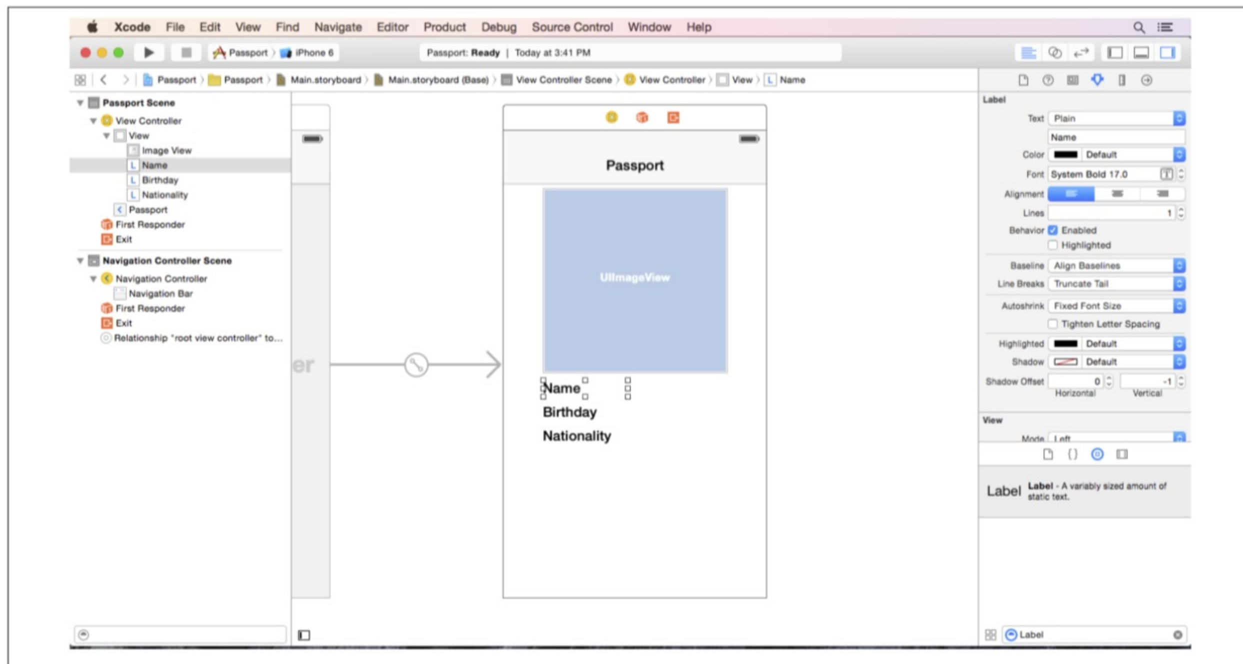
X一栏输入48，Y一栏输入70，Width一栏输入225，Height一栏输入225。然后不勾选Autoresizing中的两个相交的剪头（见图5-14）。



然后拖一个Label控件，放在Image View的左下方，然后再放两个Label控件。选中这三个Label控件，打开Attribute Inspector，在font一栏中有个带有字母T的方格，点击这个方格，弹出菜单，选择System Bold（见图5-15），点击Done。

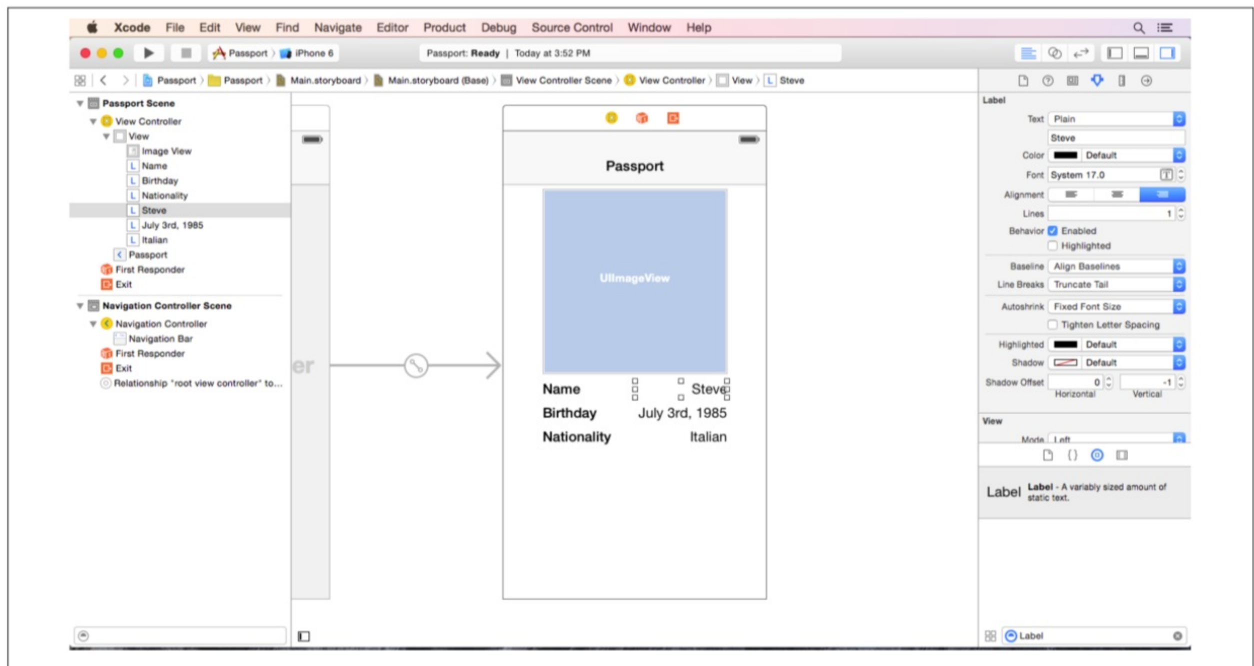
> Exercise: Passport | Page 151

把每个Label的宽调整到100pts以上，然后把3个Label控件的文字改为：Name，Birthday，Nationality（见图5-15）。

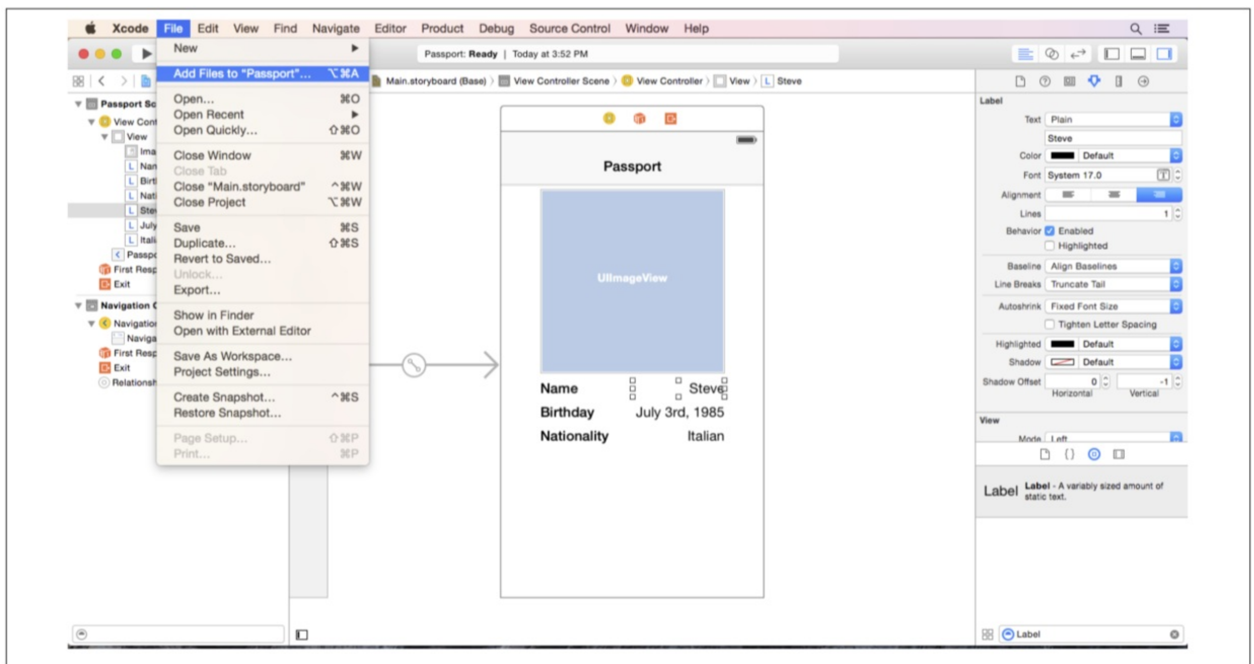


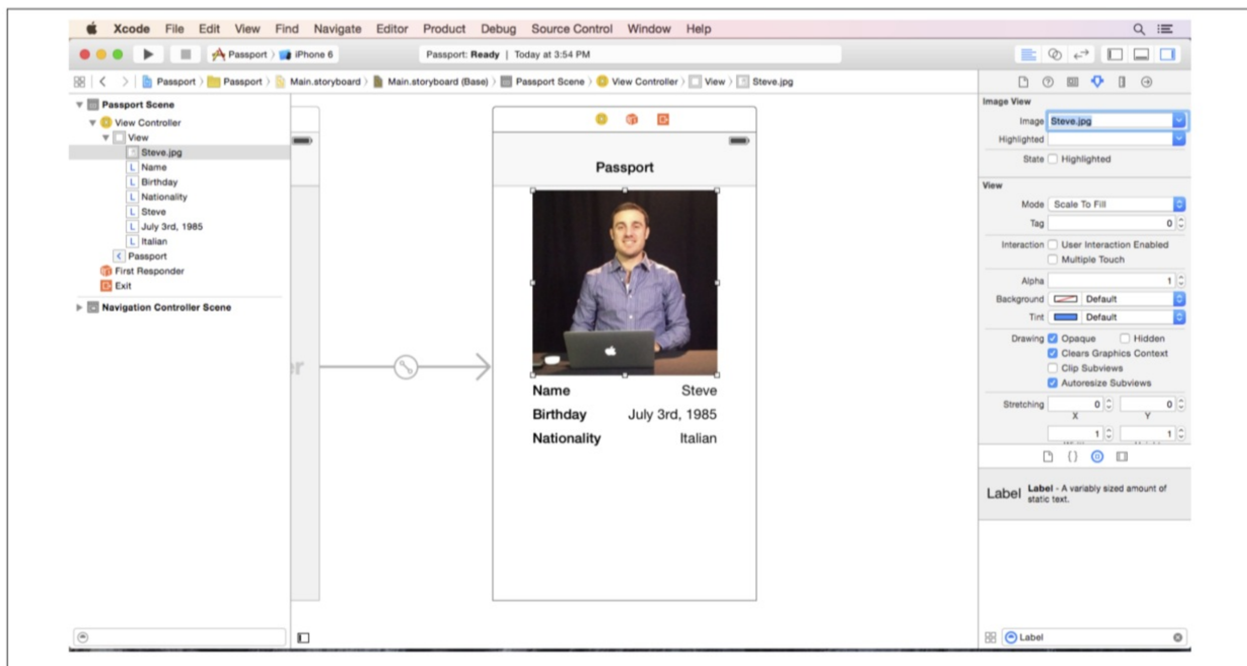
> Page 152 | Chapter 5 : Building Multiscreen Apps

然后再添加3个Label，放在右侧，对齐方式改为右对齐。在Attribute Inspector中，有个Alignment属性，可在这里找到右对齐的图标，点击图标即可修改为右对齐。宽度调整到1100pts以上，然后双击修改文字（见图5-16）。



找到一张你自己的照片，然后点击顶部菜单File -> Add Files to Passport（见图5-17）。找到你的照片，然后勾选Copy Items if needed，选择图片文件，点击Add。回到Storyboard中来，点击Image View，然后点击Attribute Inspector中的Image下拉菜单，选中刚刚添加的照片（见图5-18）。

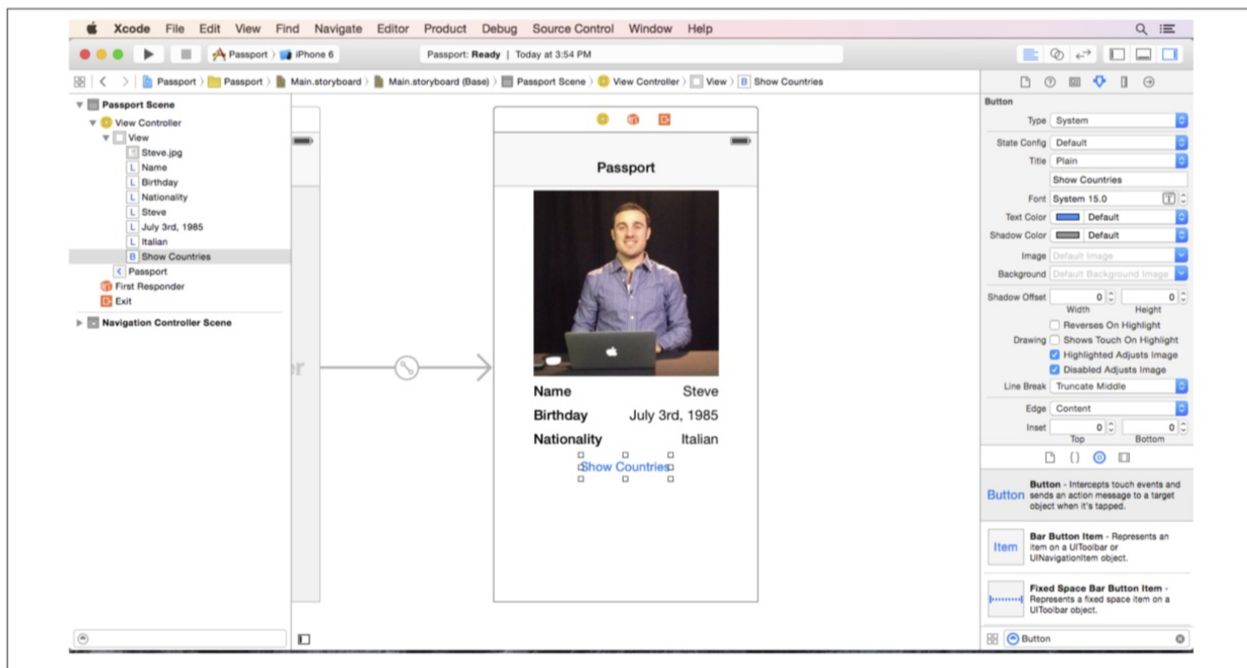




Exercise: Passport | Page 153

图片可能看起会有拉伸变形，点击Attribute Inspector中的Mode下拉菜单可以修复这个问题。选择Aspect Fill，这样会保证图片的宽高比不会变化，现在Image View中的图片看起来好多了。

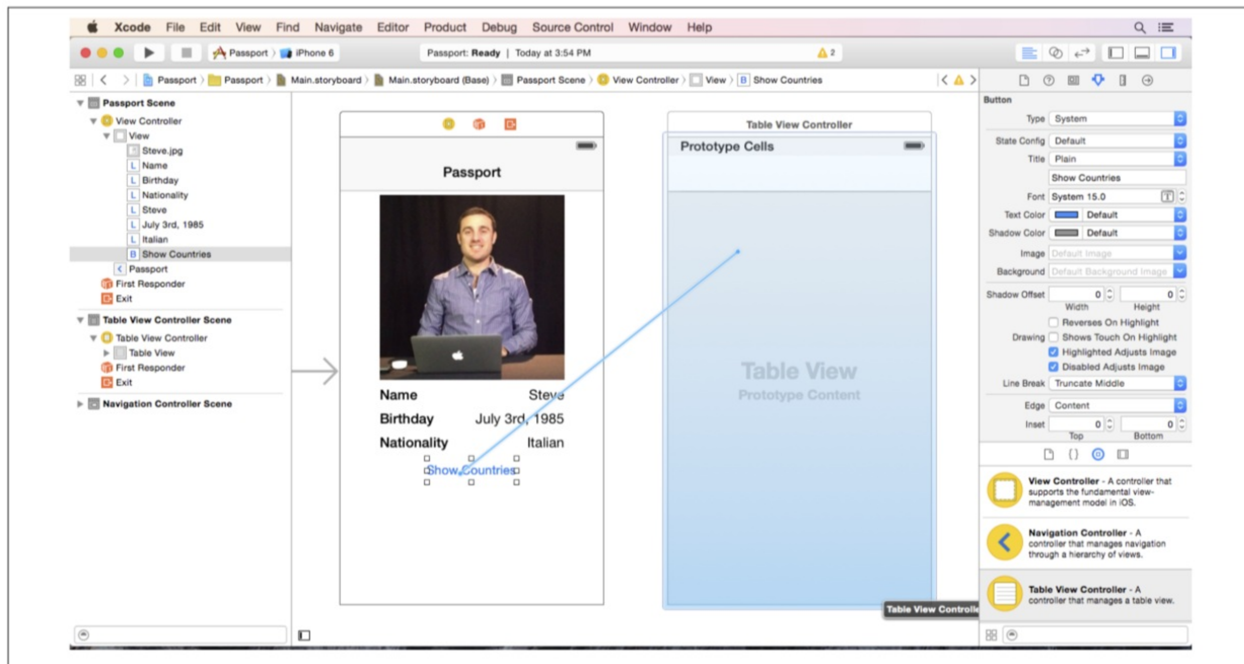
从Object Library中拖拽一个Button控件，放在个人信息的下方（就是3+3个Label控件的下方）。双击Button控件，命名为Show Countries（见图5-19）。当点击这个Button时，App会展示一个国家清单。



Page 154 | Chapter 5 : Building Multiscreen Apps

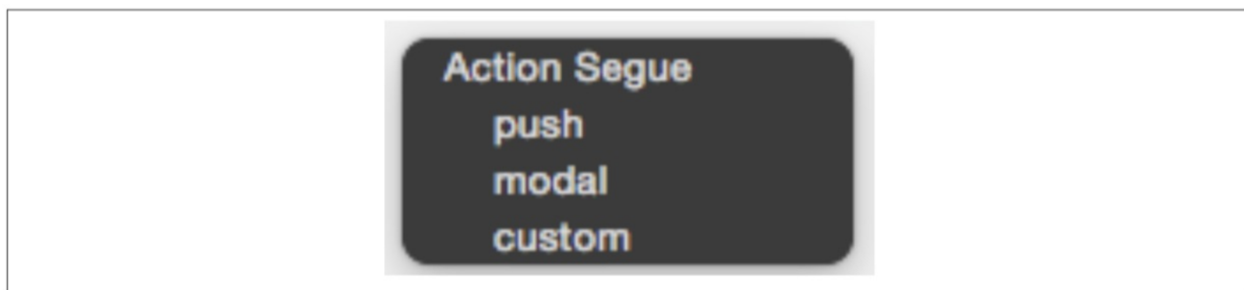
这个国家清单是由table view来处理，苹果提供了UITableViewController这个控制器来专门与table view视图协调工作。从Object Library中拖拽一个Table View Controller，放到Passport Scene旁边。

接着选中Show Countries这个Button控件，同时按住Control键，鼠标拖拽到table view controller上（见图5-20），然后松开鼠标。



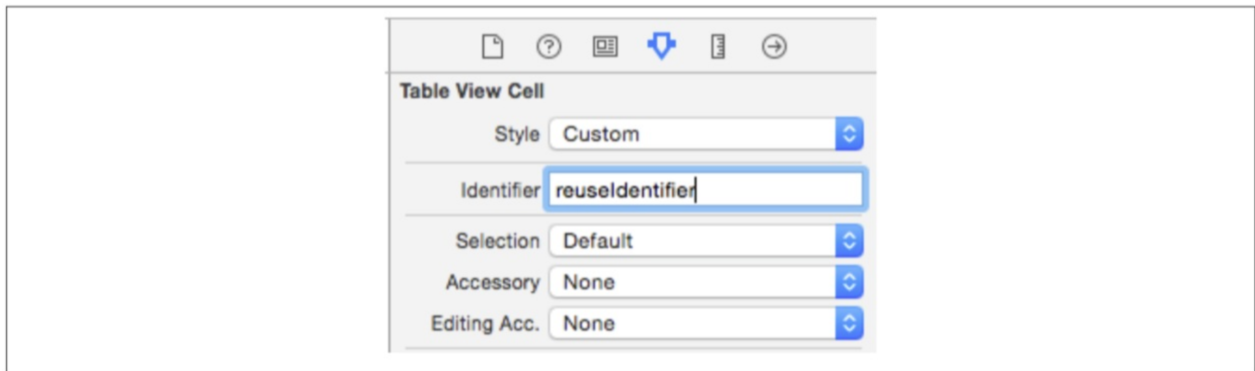
Exercise: Passport | Page 155

这时会弹出一个窗口（见图5-21），选择push，然后在两个Scene之间出现了一条线。

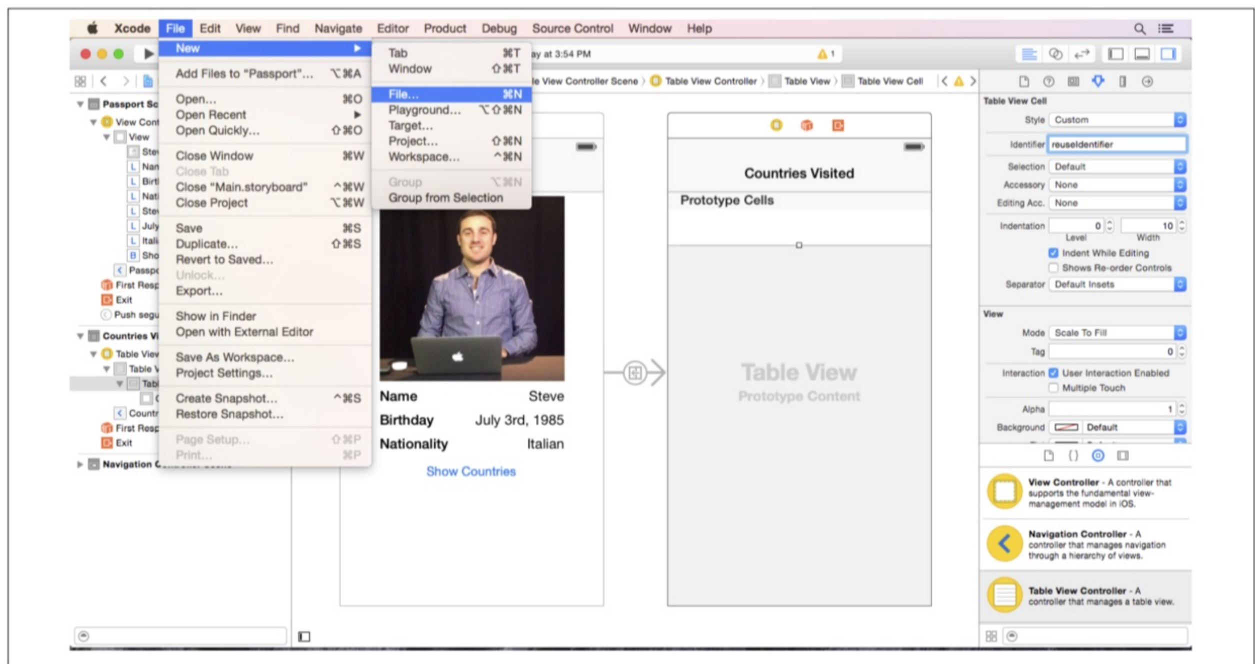


而且Table View Controller Scene中有了navigation bar，双击这个navigation bar然后输入Countries Visited。

然后点击Prototype Cells下方的空白方格，接着打开Attribute Inspector，在Identifier后面输入reuseIndentifier（见图5-22）。

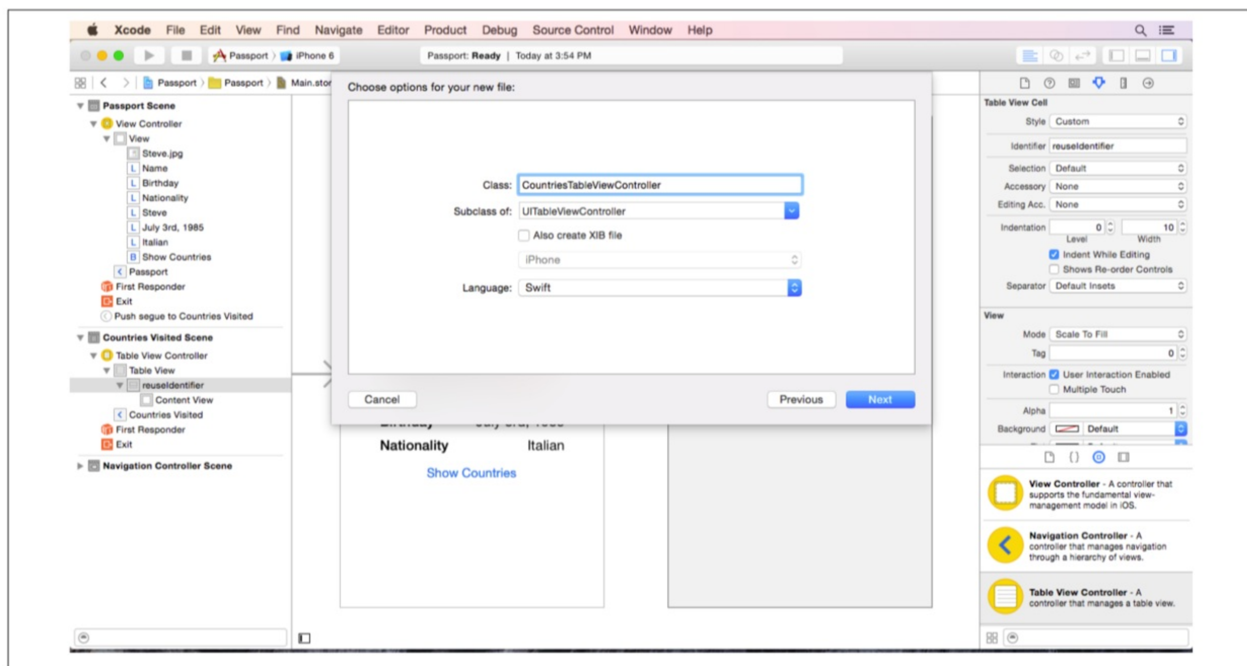


这个App的storyboard部分已经完成了。现在需要创建table view controller文件。选择顶部菜单的File -> New -> File（见图5-23）。确保Source是在iOS下，选择Cocoa Touch Class，点击Next。



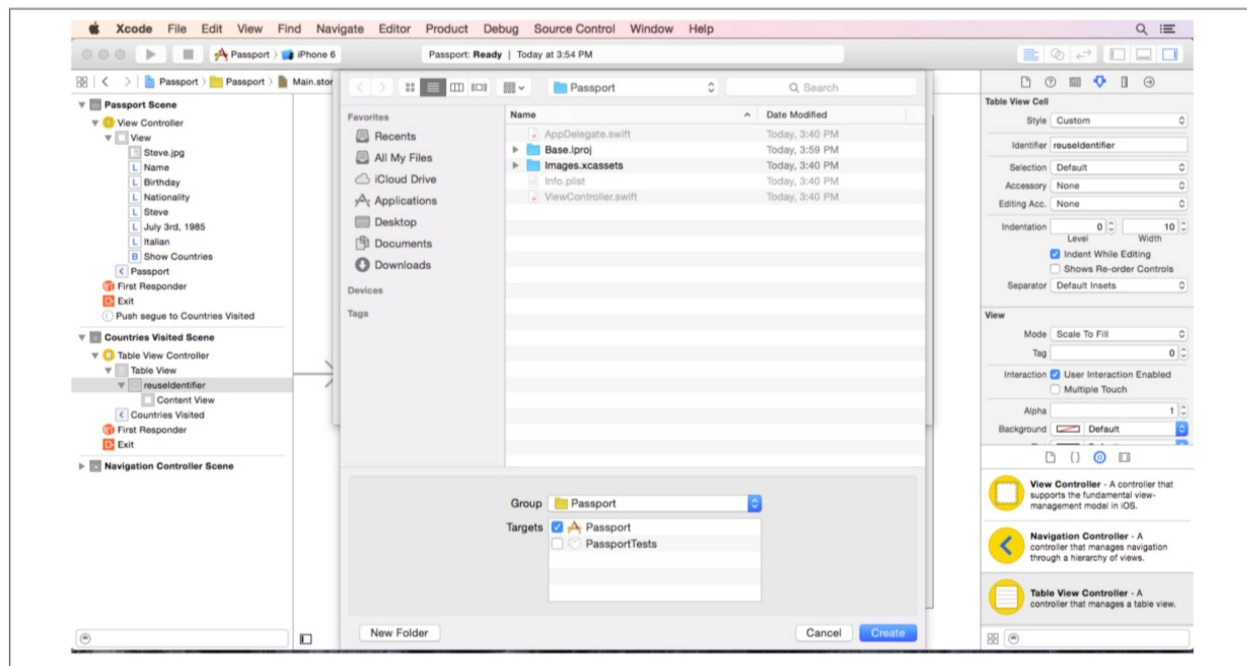
> Page 156 | Chapter 5 : Building Multiscreen Apps

Subclass of一栏选择UITableViewController，Class一栏输入CountriesTableViewController，不勾选Also create .xib，语言是Swift（见图5-24）。点击Next。



Exercise: Passport | Page 157

然后选择文件保存地点，已经自动选择了当前工程所在的文件夹，确保Passport文件夹在最上方（见图5-25），然后点击Create。



刚刚创建的*CountriesTableViewController.swift*文件已经自动在Xcode中打开了。

然后选中所有绿色的代码，删掉它们。注意不要一不小心删掉最底部的那个右大括号。viewDidLoad和didReceiveMemoryWarning中的绿色代码也删掉。然后隐藏Inspector，打开Project Navigator。

把鼠标光标放到class CountriesTableViewController: UITableViewController这行代码的下方，我们在这里创建变量属性，去过的国家应该是存储在数组当中，任何方法都可以访问数组，请输入下列代码：

```
var countries = ["Italy","Norway","England"]
```

numberOfSectionsInTableView方法确定table view中显示多少个section，把数字改为1，删掉这行//#warningPotentially incomplete method implementation，见下方：

```
override func numberOfSectionsInTableView(tableView: UITableView) -> Int {
    //Return the number of sections.
    return 1
}
```

> Page 158 | Chapter 5 : Building Multiscreen Apps

在 numberOfSectionsInTableView方法下面，还有一个numberOfRowsInSection方法，numberOfRowsInSection这个方法确定table view中显示多少行，把数字改成3，删掉这行//#warningPotentially incomplete method implementation，见下方：

```
override func tableView(tableView: UITableView, numberOfRowsInSection section:Int) -> Int
     //Return the number of sections.
    return 3
}
```

把鼠标光标放到这两个方法下面，然后输入tableView，会自动出现一些代码，这是自动补全功能，帮助你更快的输入代码，找到下面这行代码：

```
tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITabl
```

然后敲击回车，删掉代码占位符。

tableview中的每一行都会调用cellForRowAtIndexPath方法，在这个方法中我们创建UITableViewCell，然后每一行都会显示我们想显示的内容。在方法中添加下列代码：

```
let cell = tableView.dequeueReusableCellWithIdentifier("reuseIdentifier",  forIndexPath:
```

这行代码创建了一个名为cell的常量，等号后面是tableView中可回收的cell，这个方法用来检查使用已经有了一个可回收的cell，且其identifier为reuseIdentifier。如果有这个cell，那么让cell可回收；反之，创建一个新的cell。接着添加下列代码：

```
var country = countries[indexPath.row]
```

这行代码把国家名字赋值给了country这个变量。table view中的每一行都会调用cellForRowAtIndexPath这个方法，每次被调用，indexPath变量会更新，然后提供最新的section和row number。例如，cellForRowAtIndexPath第一次被调用的时候，是第一section第一行，接着是第一section第二行，然后是第一section第三行，继续直到遍历完所有的section所有的行。indexPath.row用来提取数组中的第一个第二个和第三个国家。

接着添加下列代码：

```
cell.textLabel.text = country
```

这行代码把coutry变量的值赋值给了UITableViewCell的text属性，这样，国家名字将会展示在每行中。

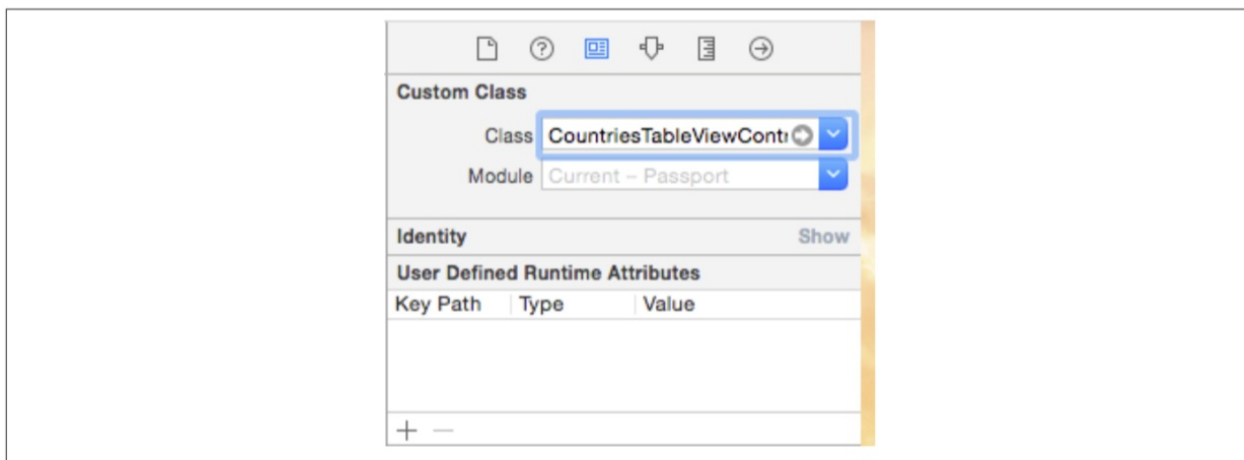
> Exercise: Passport | Page 159

添加最后一行代码：

```
return cell
```

应用几乎快要完成了，然而还需要修改Storyboard中的一些设置。打开Main.storybaord，然后打开Inspector。

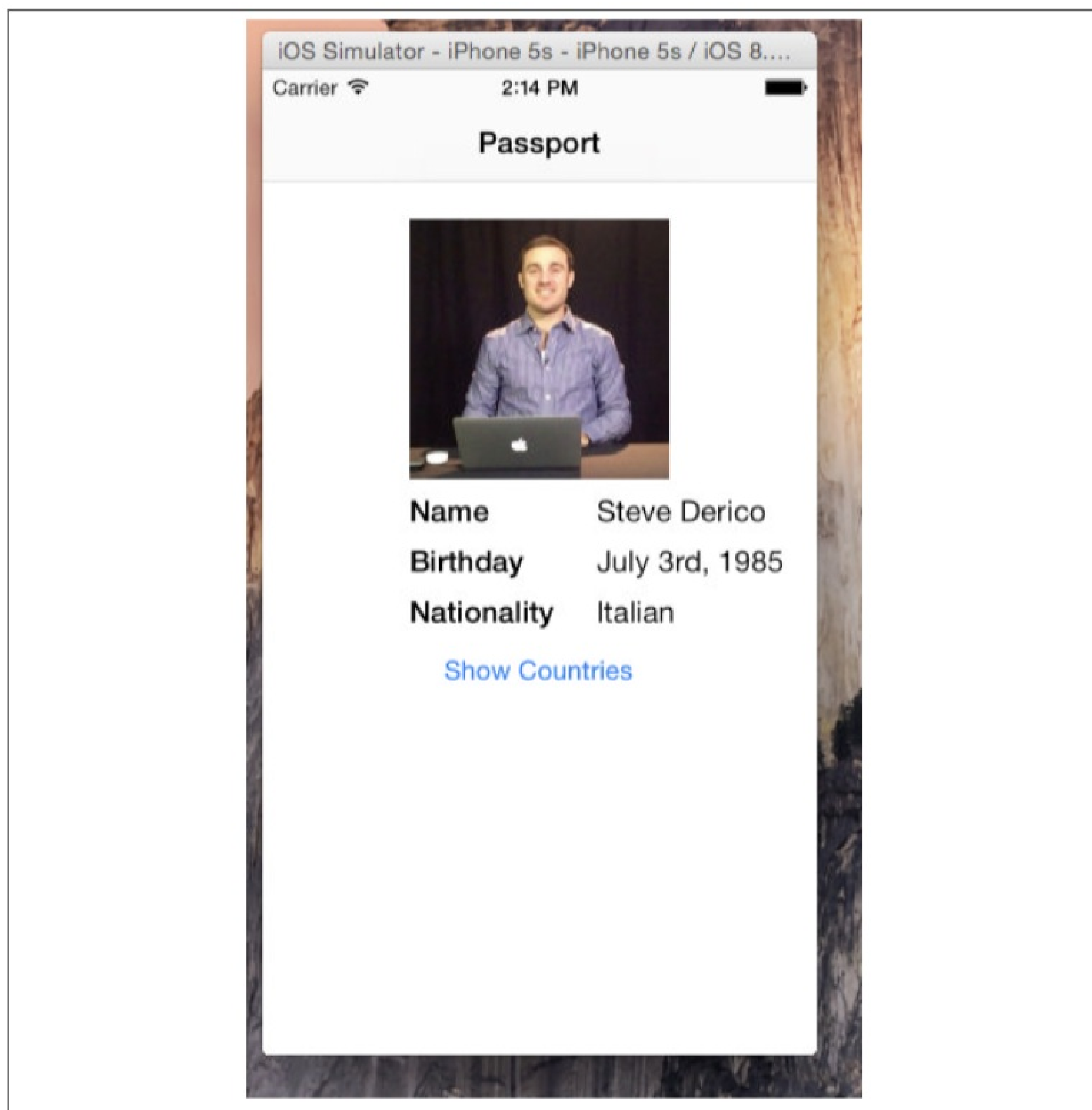
接着点击Table View Controller Scene，接着选中Table View Controller Scene上方的黄色圆圈，选中后一个蓝色框会包裹住黄色圆圈，打开Identity Inspector，从左数第三个图标，看起来像是张报纸，在Class一栏中选择CountriesTableViewController（见图5-26）



这个操作会把*CountriesTableViewController.swift*和Table View Controller Scene连起来。现在应用已经编写完成了，保存你的操作，然后点击左上角的Paly按钮（或者快捷键Command+R）。

App启动后会展示姓名生日国籍和照片（见图5-27）。点击Show Countries按钮，navigation controller会自动push一个新的view controller到最前面。

> Page 160 | Chapter 5 : Building Multiscreen Apps



每一个cell都显示一个不同的国家（见图5-28）。左上角的back（返回按钮）是自动生成的，文字显示的是之前界面的title，如果之前的界面没有title，就会显示Back。点击Back这个按钮。

> Exercise: Passport | Page 161



个人挑战： 如果你想让这个App能够展示5个国家，你需要做哪些改动呢？展示10个呢？或者1000个国家？

如果App没有按照你想要的结果运行，或者程序有了错误或警告，不要太担心，学习的最佳方式就是试错，熟能生巧，到我们的网站上下载示例代码，对比一下代码，多试几次，直到搞定这个程序为止。

Page 162 | Chapter 5 : Building Multiscreen Apps

第六章：下一步：调试，文件和App图标

在这一章节中，你将学会如何调试你的代码。调试（Debugging）是开发者非常重要的工具，会调试代码可以为你节省几百个小时的时间，同时，你还将学会如何阅读苹果公司的开发文档（Apple's documentation），这个文档就是苹果公司提供的百科全书，这里有你想知道的所有答案，最后你还将学会如何给你的App增加图标和启动页面。让我们继续增加你的知识吧。

为什么要调试？（Why Debugging?）

开发软件的大部分工作都是报错然后解决问题。问题（Issues），也叫做bugs，就是你的App没有按照你预想的来运行。分析问题筛选原因检修代码的过程叫做调试（Debugging）。苹果公司甚至在Xcode中提供调试工具套件，帮助你完成调试工作。

调试（Debugging）是开发工作的一个步骤，不管你开发过多少个App，你还是需要调试代码。和微软的Word中的拼写检查相似，不会因为你已经写过500多篇论文，就表示你不会犯语法错误或者拼写错误。当你需要调试代码的时候，不要沮丧，记着，调试只是一个必须经过的步骤，能够提高你App的质量。

一般会有两类问题：compile time（编译时遇到的问题）和runtime（运行时遇到的问题）。

编译时遇到的问题（Compile-Tiem Issues）

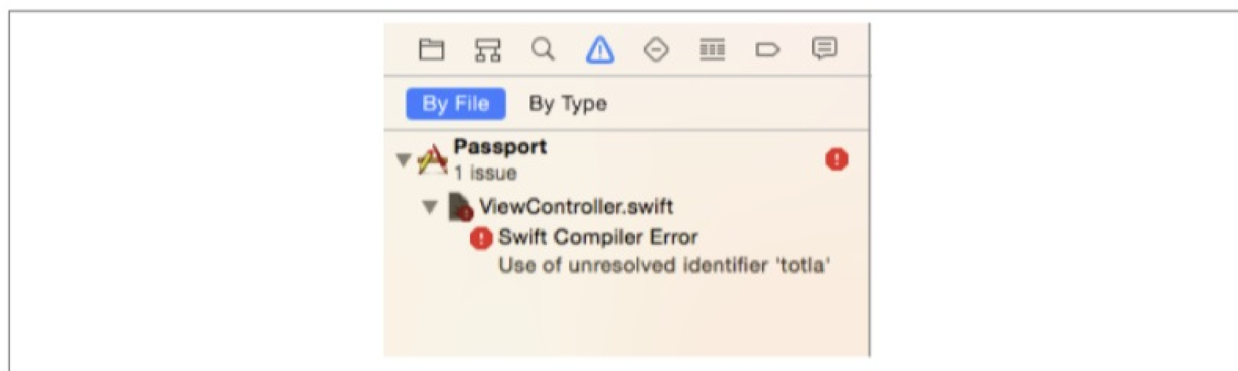
要理解编译时遇到的问题（compile time issues）这个概念，你先要理解Xcode背后运行机制。Xcode读取swift代码然后翻译成数字0和数字1，以便让iOS模拟器能够理解。把swift编译成0和1过程，叫做compiling（编译），也就做building。

> Page 163

当你的代码正确无误时，就能够通过编译，如果代码有错误，就无法通过编译，也就无法在iOS模拟器中运行。点击屏幕左上角的Play按钮开始编译。如果编译成功，程序就会在模拟器上运行，如果失败了，Xcode上会出现一个红色的报错提示。例如，看一下下面这个代码：

```
var total = 100 + 40
var half = totla/2
```

这时第二行代码会出现一个编译错误，因为没有一个叫totla的变量，所以引起一个错误，编译失败（见图6-1）。



Xcode会告诉你造成错误的原因。例如：

```
var total = 100+40
var half = totla/2
 //Use of Unresolved Identifier 'totla'
```

这些原因一开始听起来有些含义模糊神秘，不过过一段时间， 就会习惯了。在这个例子中，Xcode说有个identifier，也就是有个变量，被使用了，但是没有找到这个变量。翻译一下，就是说“你正在使用一个不存在的变量”。变量 `totla` 不存在是因为出现了拼写错误，把totla改成total，就会解决问题：

```
var total = 100+40
var half = total/2
 //Build Successful - No Errors
```

遇到bug不要气馁，有些bug可以几分钟内就解决掉，有些几个小时，有些甚至需要几天来解决。但是记住，犯错出现问题和解决问题才是最好的学习方式。再也没有比看一眼bug立马解决更好的了。保持积极心态，和一点点坚持， 就能越走越远。

> Page 164 | Chapter 6 : Next Steps: Debugging, Documentation, and App Icons

错误（Errors）

编译错误造成的原因可能是句法错误，获取属性值错误，或者是匹配错误类型。这些错误都会把Xcode发现然后用红点标注在出现错误的代码左边，Xcode的编译错误检查机制和微软Word的语法拼写检查机制相似，有过有错误，都会高亮显示，提出建议解决方法。

Xcode具有提供修改建议的能力，叫做Fix-It。如果出现了修改建议，出现的红点里的叹号就变成了一个白点，点击这个白点，弹窗一个菜单，呈现问题已经可能的修改方法。点击Fix-It选项，提议的修改方式就会增加到你的代码中去。

警告（Warnings）

在某些情况下，Xcode会怀疑某个地方有问题，但是仍然会运行程序。这时，Xcode就会提出警告（warnings）。有可能是变量的类型没有声明，有可能是某个变量没有被用到。如果出现了一个警告，会在响应代码左边出现黄色三角符号。点击这个黄色三角符号，显示可能出现的问题。大多数情况下，警告都是需要引起你重视的，最好和处理错误一样，处理警告的问题。

当你的App完成编译后，错误和警告的数量会在Xcode顶部出现。如果有错误，Xcode就会停止编译，然后出现编译失败的通知（见图6-2）。如果只有警告没有错误，那么Xcode会继续编译，出现编译成功的通知。所有的错误和警告都会整理到一个地方：the Issue Navigator。



Figure 6-2. Build Failed notice

> Compile-Time Issues | Page 165

Xcode左侧窗口的Issue Navigator目前可用，点击Project Navigator工具栏从左起第四个按钮（三角形带一个叹号的按钮），就能打开Issue Navigator，打开后，所有的问题、错误和警告都会展示出来，点击其中一个问题，Editor就会打开相关的文件，高亮标出带问题的代码，Issue Navigator也提供问题的描述，能够更简单的找到问题所在。

运行时的问题（Runtime Issues）

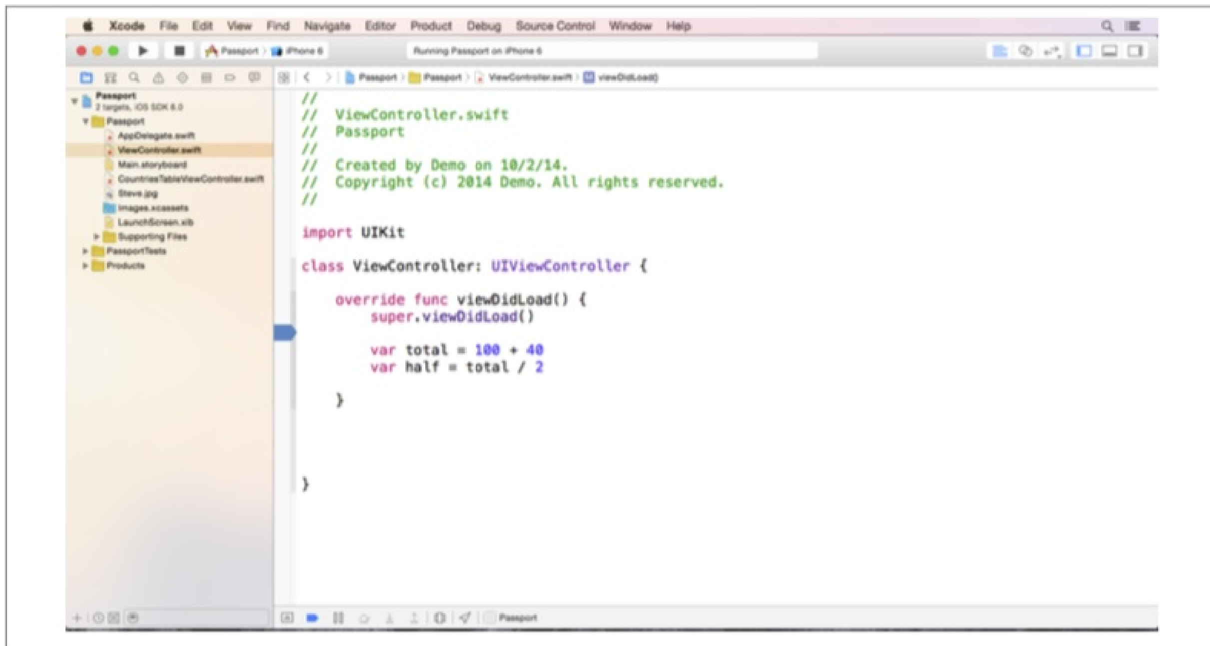
Runtime就是应用正在运行界面启动。例如，运行时的问题意味着在点击某个按钮时，应用崩溃（crash）。比起解决编译时的问题，解决运行时的问题的难度更大，因为我们不太容易确定到底是哪行代码出了问题，这时候断点（Breakpoints）会帮助你。断点（Breakpoints）允许开发者临时暂停程序运行，可以一行一行的检查代码。一行一行运行的过程也就做stepping（步进）。stepping（步进）代码就像是看即时回放的慢镜头，可以一帧一帧地观看到底发生了什么。

当程序运行时遇到了无法解决问题，就会发生崩溃（crash），App停止工作用户无法使用。

断点（Breakpoints）

Editor左侧叫做gutter（栏距），在gutter上点击某行代码所在位置，就能产生一个断点（Breakpoints），在gutter上会出现一个深蓝色三角，表示在这行代码产生了一个断点（Breakpoints）（见图6-3），当程序运行到这行代码时，就会结束。

> Page 166 | Chapter 6 : Next Steps: Debugging, Documentation, and App Icons



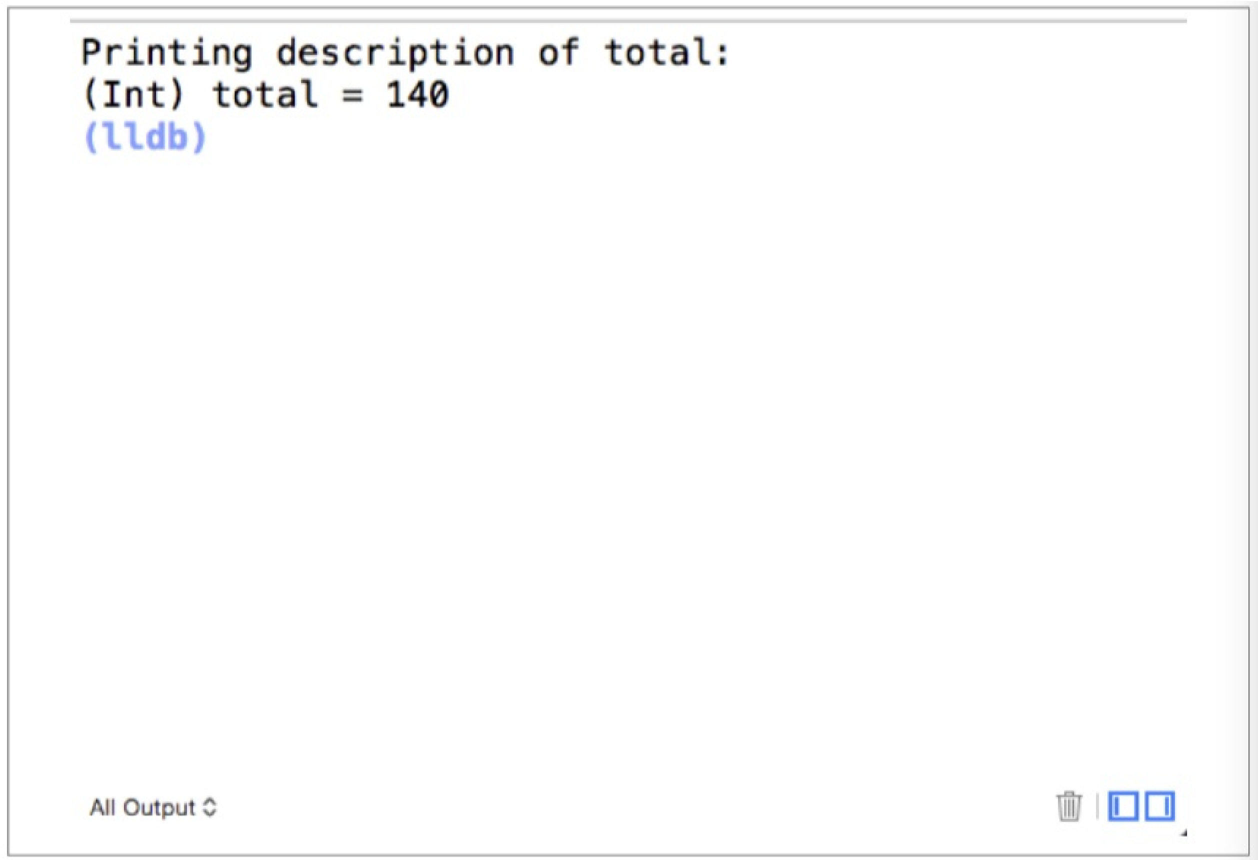
设置断点（Breakpoints）后，程序运行到此行代码结束，然后显示Debugger（调试器）。

在底部显示出一块面板叫做debug area（调试区域）（见图6-4）。调试区域有两部分：变量界面（variables view）和控制台（console）。变量界面在左边，显示当前方法中存在的变量。



右击一个变量，然后选择“Print description of variable”，把变量的细节信息显示到控制台（console）上，控制台（console）在右侧，显示开发日志（developer log）（见图6-5），可以监视App的运行情况。

Runtime Issues | Page 167



`println()` 方法（swift2.0之后语法已经修改为 `print()` 了，请注意）可以将信息显示到控制台上：

```
println("Button Tapped")
```

`println()` 方法由直接把字符串的内容显示到控制台中。

使用调试器（Using the Debugger）

图6-6是变量界面中的调试工具栏，表格6-1显示的是调试工具栏中从左到右每个按钮的作用。



Page 168 | Chapter 6 : Next Steps: Debugging, Documentation, and App Icons

表格6-1 Debug toolbar 调试工具栏

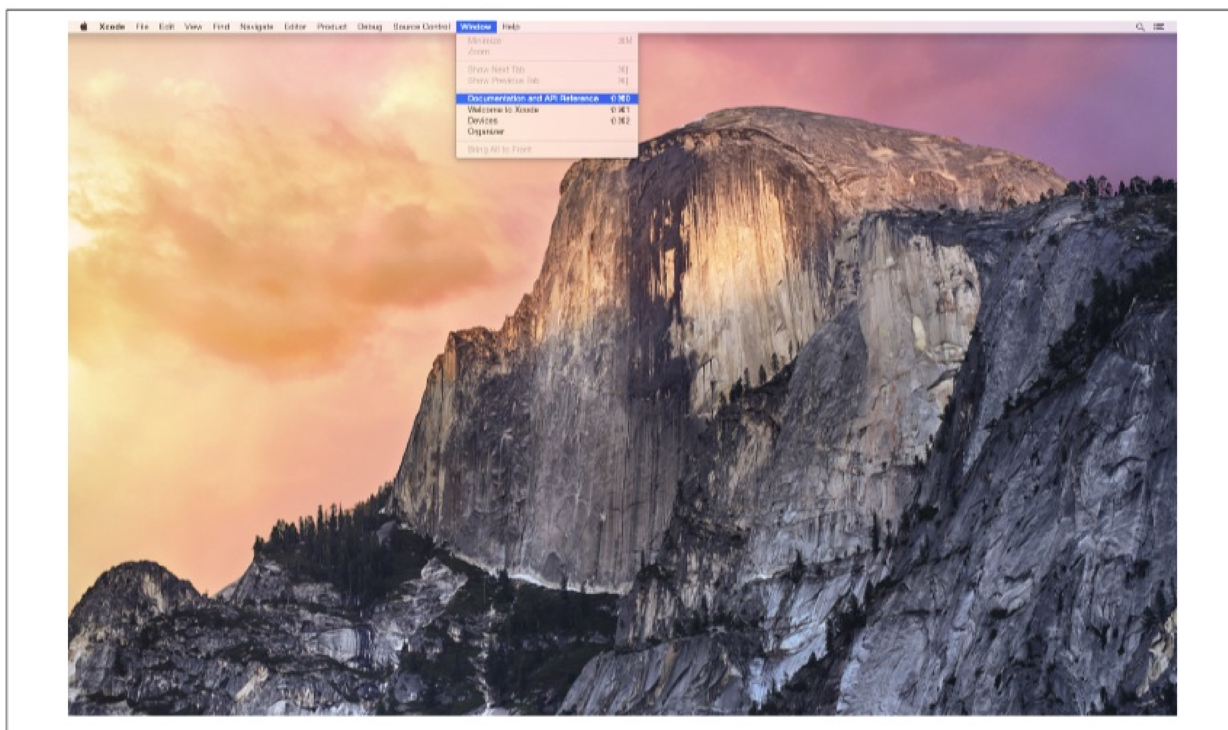
Button 按钮	Description 作用描述
Show/Hide 显示/隐藏	Debug area 调试区域
Enable or Disable Breakpoints开启或关闭断点功能	Breakpoints 断点
Continue 继续	Resumes app back to normal state or until next breakpoint 让App继续运行直到遇到下一个断点
Step Over 下一步	Executes next line of code, like a frame-by-frame forward button检查下一行代码
Step Into 进入	Moves the debugging into a method, if applicable 如果可行，进入方法中去调试
Step Out 退出	Moves the debugging out of a method, if applicable 如果可行，退出方法
Debug View Hierarchy 调试界面等级	Displays view for debugging interface elements and view layouts 展示调试界面元素的等级
Simulate Location 模拟城市地点	Selects a city to simulate GPS coordinates 选择一个城市然模拟GPS定位
Method Name 方法名字	Shows the current method being called 显示当前被调用的方法的名字

Step over按钮使用最为频繁，Step over可以像即时重放那样让代码一行一行执行，控制台（console）也比较常用，可以确认一个事情发生或者日志记录。

文档（Documentation）

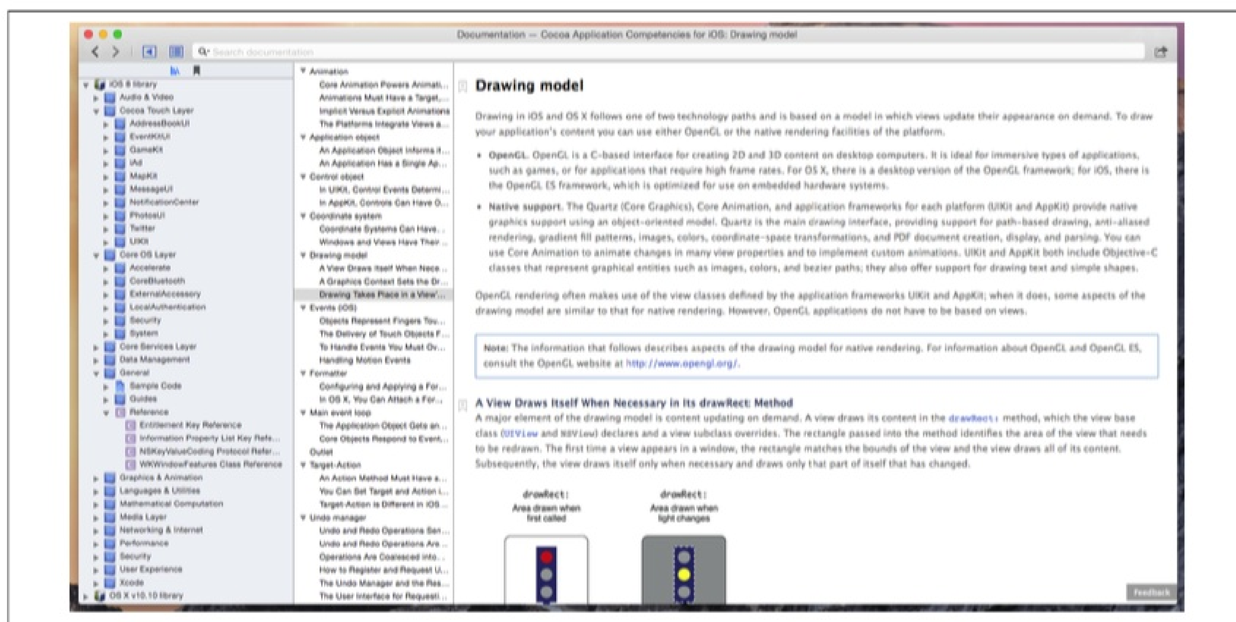
文档是任何编程语言的圣杯，文档描述了方法、类、属性、和变量如何工作，是开发者最好的朋友，提供需要的细节描述，像方法的参数类型，返回值之类的细节。苹果公司把大量的时间和资源投入到编写文档工作中，如果你有疑问，首要就是要查询文档。

点击最上方的Window按钮，然后点击“Documentation and API Reference”（见图6-7），就打开了文档，当你编程的时候，最好能够一直打开文档。



Documentation | Page 169

文档窗口看起来有些像是网页浏览器（见图6-8）。在输入框中输入你想查询的关键词，出现结果列表。输入框左侧的三条线的按钮是内容窗格表格，这个窗格会显示与某个主题相关的所有内容，快速查询某部分内容。它左侧的按钮是显示Library Navigator，浏览文档的目录内容，再往坐的两个前进和后退按钮功能就是前进和后退了。



Page 170 | Chapter 6 : Next Steps: Debugging, Documentation, and App Icons

最后，右侧的主窗口显示选中的类、注释、参考的具体内容。拿类举个例子，首先会显示类的一个简单描述，然后列出这个类里所有的方法和属性，每个方法属性都会有描述和例子，类会描述每个参数和返回值，当显示这个方法或属性是在哪个版本时推出的（created in iOS

3.0就是在iOS3.0的时候才有的这个方法）。

示例代码（**Sample Code**）

文档不仅仅是显示指南文件，还提供实例代码和示例工程，研习示例工程是学习苹果技术的好方法。比如，你打开Library Navigator然后打开iOS8 library，打开User Experience然后点击Sample Code选中HelloWorld，这个示例工程是展示如何创建一个简单的程序，来接收用户输入然后显示输入信息，点击Open Project查看示例代码。（不知道为啥我打不开啊，可能是因为这个示例工程是用OC写的，而我用的是swift？能打开的好心告诉我一下吧~）

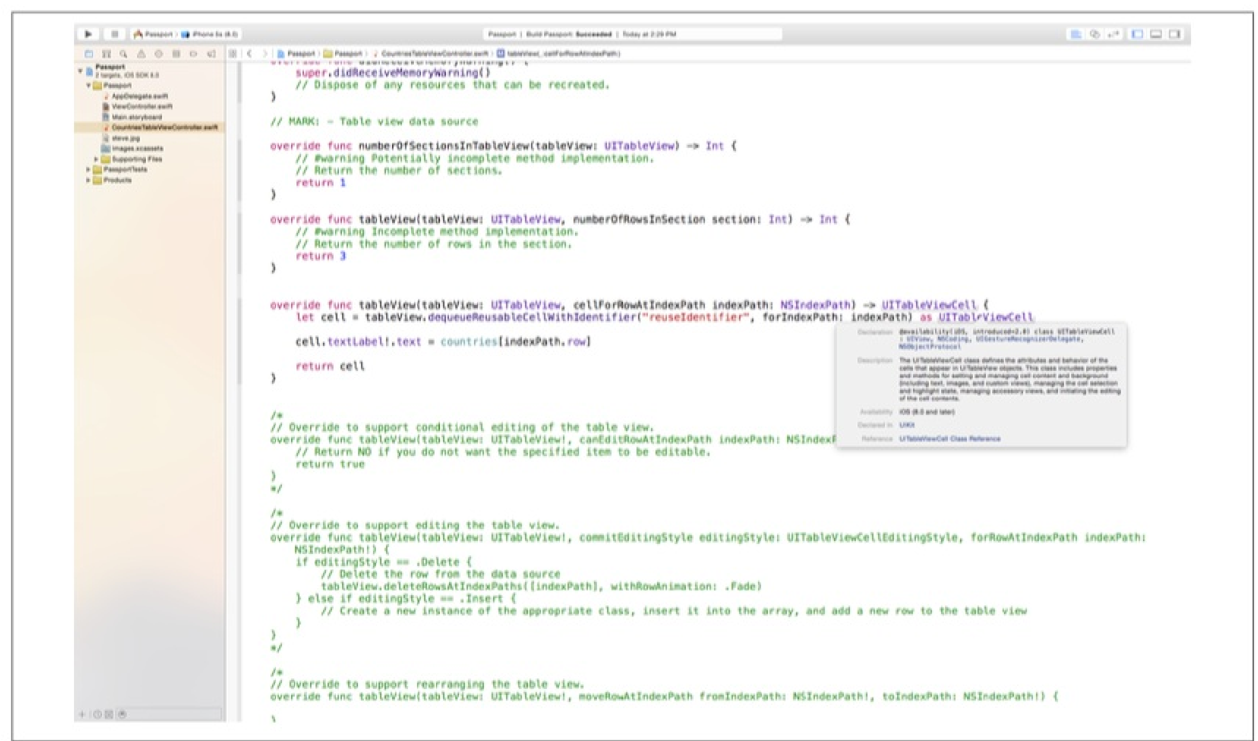
如何最有效地使用文档（**How to Get the Most Out of Documentation**）

每一个优秀的程序员都会花时间精力来学习文档，这是成为高手的最好方法，既然文档这么重要，那么就介绍一下其中的一些你应该知道的小技巧吧。

如果你对某个类、关键词或者方法有疑问，把你鼠标移到上面，然后按住Option键，鼠标点击，这时就会出现一个弹出框Quick Help（见图6-9）。显示说明、解释和跳转到参考文档的链接。

> Documentation | Page 171

点击参考链接，跳转到文档。或者按住Option然后双击想要查询的项目，就不会出现弹出框Quick Help而是直接跳转到文档。当然了，还可以使用Command+Shift+0快捷组合键直接打开文档。



文档中最有价值的部分是iOS Human Interface Guidelines（iOS人机界面交互指南）。解释了如何使用iOS中的每个界面元素，这个文档必须要读一下，如果你不遵循其中的要求，等你上传App到苹果市场时是会被拒的哦。

优秀的程序员都是非常熟悉文档的，把阅读文档当成编程的一部分，理解和学习文档的技能需要时时练，了解哪些框架和资源你目前能够使用。使用Library Navigator能够了解更多的主题。

App图标（App Icons）

App图标会出现在App应用市场，Home Screen手机屏幕上，以及Spotlight上。App 图标涉及多个设备和的很多尺寸，表格6-1是所有需要的App图标。

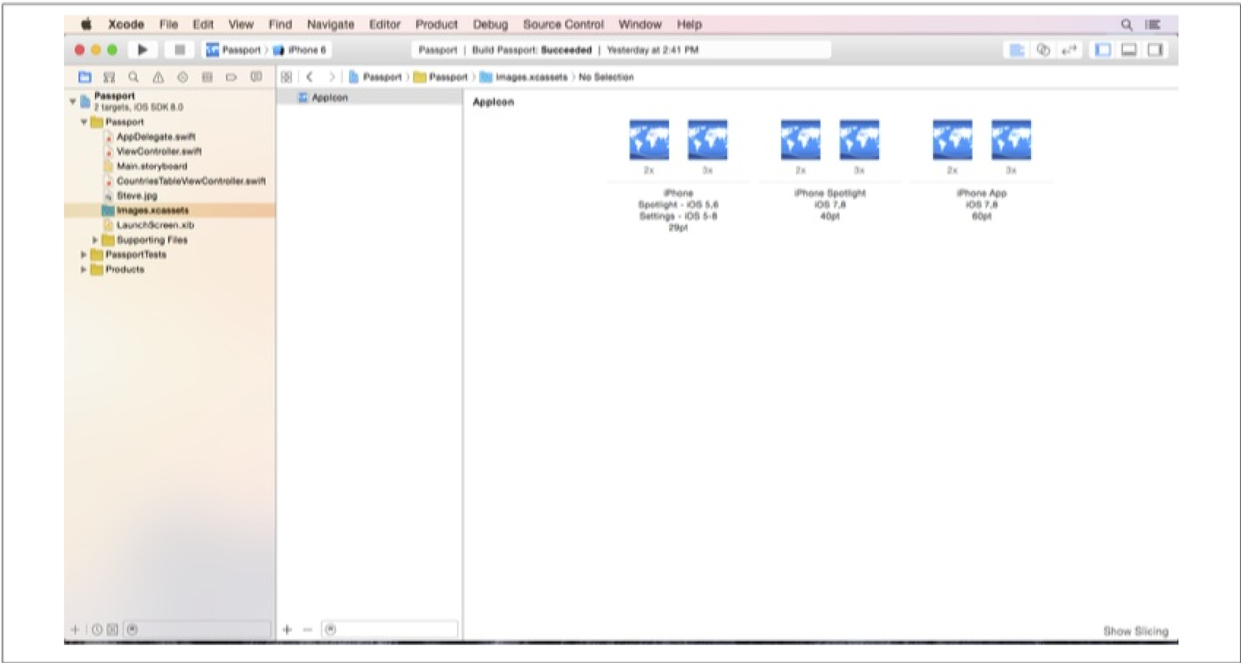
> Page 172 | Chapter 6 : Next Steps: Debugging, Documentation, and App Icons

表6-2 所有的App图标尺寸

Filename 图片名称	Image size (px) 图片尺寸	Description 描述
iTunesArtwork.png	1024x1024	App Store Listing
Icon-60@3x.png	180x180	Retina HD Home Screen
Icon-60@2x.png	120x120	Retina Home Screen
Icon-76@2x.png	152x152	iPad Retina Home Screen
Icon-76.png	76x76	iPad Home Screen
Icon-40@3x.png	120x120	Retina HD Spotlight
Icon-40@2x.png	80x80	Retina Spotlight
Icon-40.png	40x40	Spotlight
Icon-29@3x.png	87x87	Retina HD Settings
Icon-29@2x.png	58x58	Retina Settings
Icon-29.png	29x29	Settings

（我正在翻译本书版本是14年12月的，那时候还没有出来iPad pro设备） 关于Retina和Retina HD的更多信息将会在第七章中详细介绍。强烈推荐把命名时使用表格中推荐的名字，如果名字没有按照规则命名，可能会产生混淆和错误。图标做好后，就添加到Xcode中去。

想要添加到Xcode中，先要点击Project Navigator中的images.xcassets，Editor编辑器的左边会有一竖栏，选择AppIcon，出现图6-10。



App Icons | Page 173

把每个图标图片拖到相应的格子中，运行App看一下iOS模拟器中的图标是否正确，iOS模拟器启动后，点击菜单栏，选择Hardware -> Home，或者快捷键Command+Shift+H，就可以看到App图标了。

启动界面图片（Launch Image）

当App启动时，出现启动界面表示应用正在启动中，这个启动界面是由一个图片构成的。如果不设置启动界面，用户会看到一篇黑色，不确定这个应用是否启动。启动界面一般是竖屏的，然后，也可以提供横屏启动界面。大多数的启动界面和打开App后的第一个界面有些相似，Facebook的App就是这样的。

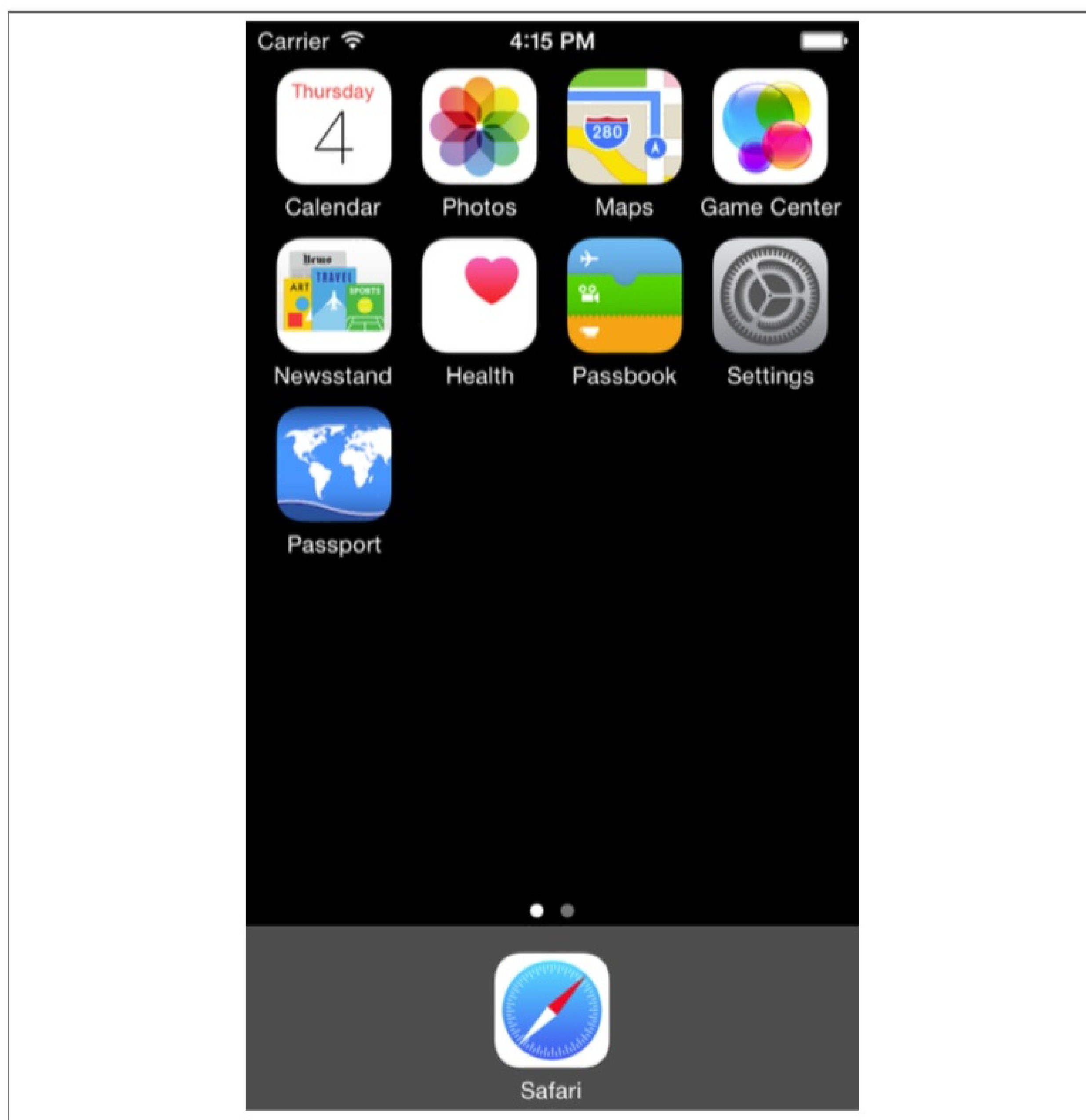
Xcode会自动为你的工程创建启动界面。LaunchScreen.xib文件用来定义启动界面，可以石笋Auto Layout自动布局，这个知识会在下一章中介绍。

在这一章中，你学会了如何调试你的代码，这样能比以前更快的解决问题，还学会了如何阅读文档，读的阅读，你的进步越快，最后，你学会了如何创建App图标和启动界面。保持这样的速度，继续加油吧~

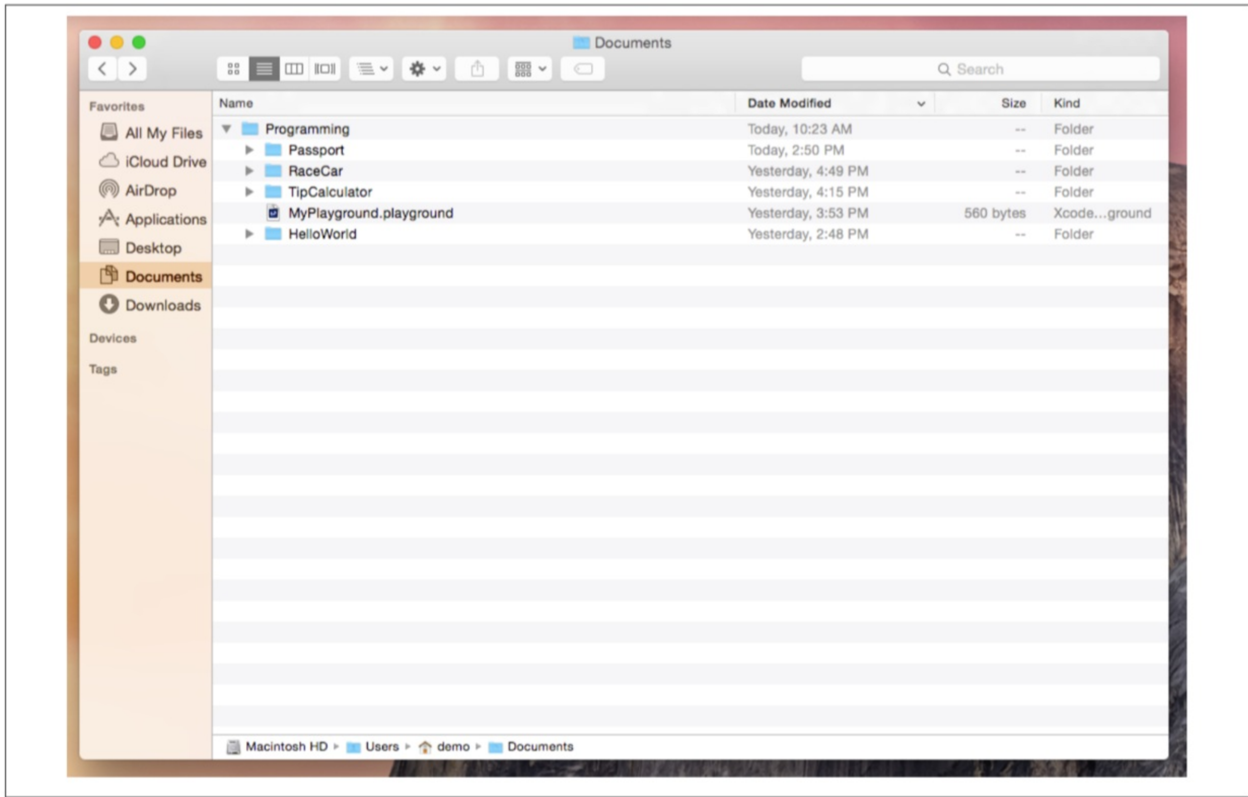
第六章练习 Expanding the Passport App - 扩展护照App的功能

这次的练习我们将扩展第五章开发的Passport App，在本次练习中，我们将全面过一遍Debugger和documentation（文档），最后，我们还会给App增加应用图标和启动界面。见图6-11。

> Page 174 | Chapter 6 : Next Steps : Debugging, Documentation, and App Icons

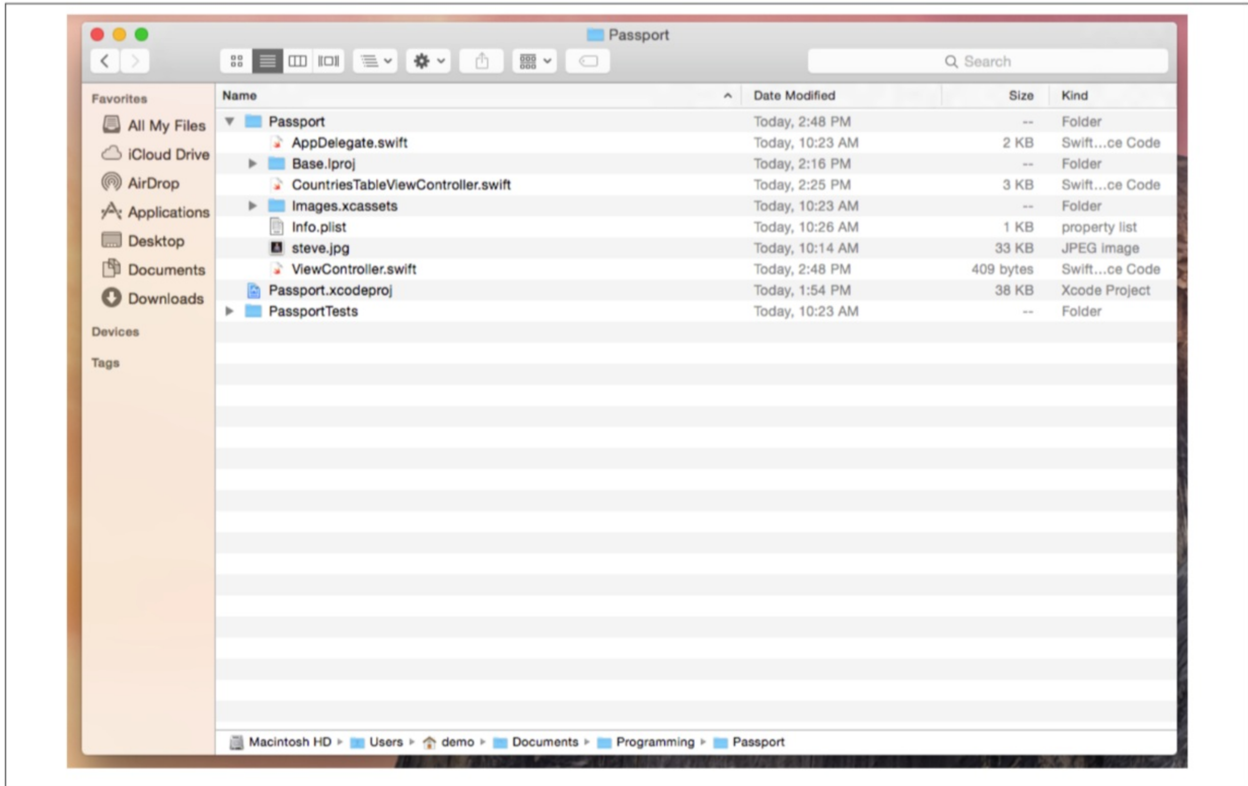


首先我们找到Passport工程文件夹（见图6-12），打开Passport文件夹，文件夹中有三部分，两个文件夹和一个*Passport.xcodeproj*文件。



Exercise: Expanding the Passport App | Page 175

见图6-13的文件，这些文件构成了Passport应用，在这里的文件层次和你Xcode中Project Navigator看到的文件层次是不一样的。

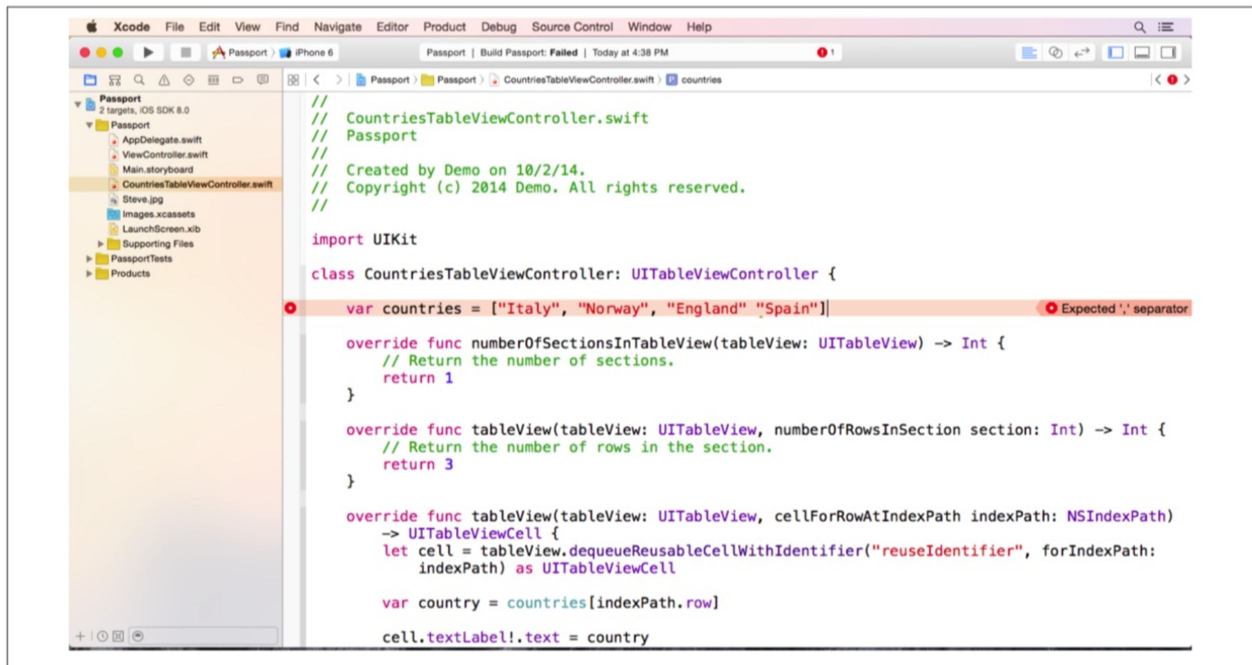


Page 176 | Chapter 6 : Next Steps : Debugging, Documentation, and App Icons

三个*.swift*文件，文件名中包含其代表的类，*Base.lproj*文件夹中有*.storyboard*文件，*image.xcassets*文件夹中保存App的图标，*Info.plist*文件用来存储App的一些细节，例如版本号，bundle identifier（Bundle ID）。双击扩展名*.xcodeproj*的文件打开工程。

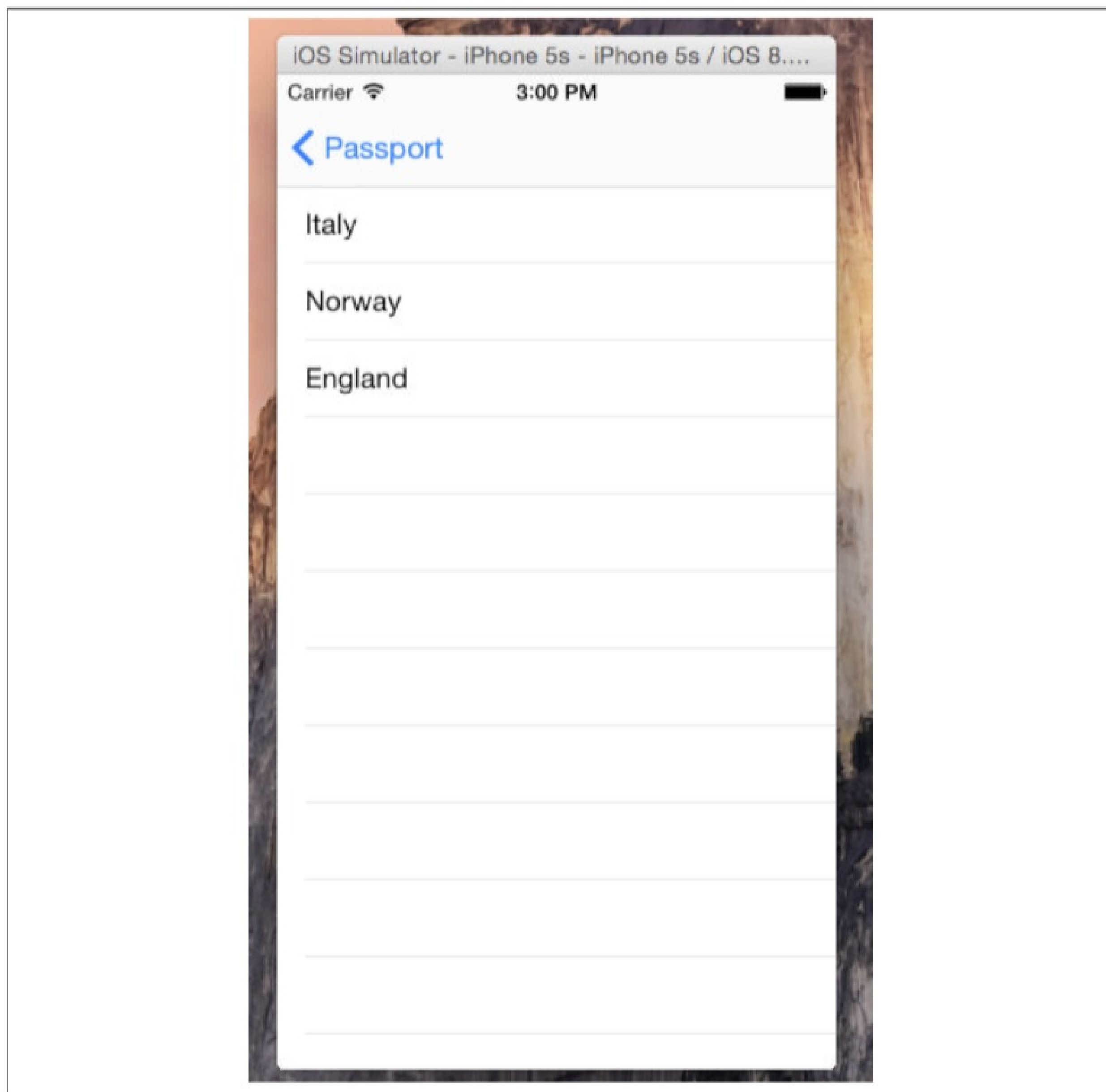
我们要在Passport App中增加国家，然后显示在国家清单中。点开*CountriesTableViewController.swift*文件，把鼠标放到数组中England后面，然后输入“Spain”，要在数组的中括号中输入，然后点击Play按钮（Command+R）。

哎呀，报错了。点击左侧的红点，然后出现Fix-It弹出框（见图6-14）。点击Fix-It，自动在数组中添加一个逗号。然后点击Play按钮（Command+R）。



> Exercise: Expanding the Passport App | Page 177

App启动后，点击Show Countries按钮，查看刚刚增加的国家（见图6-15）。

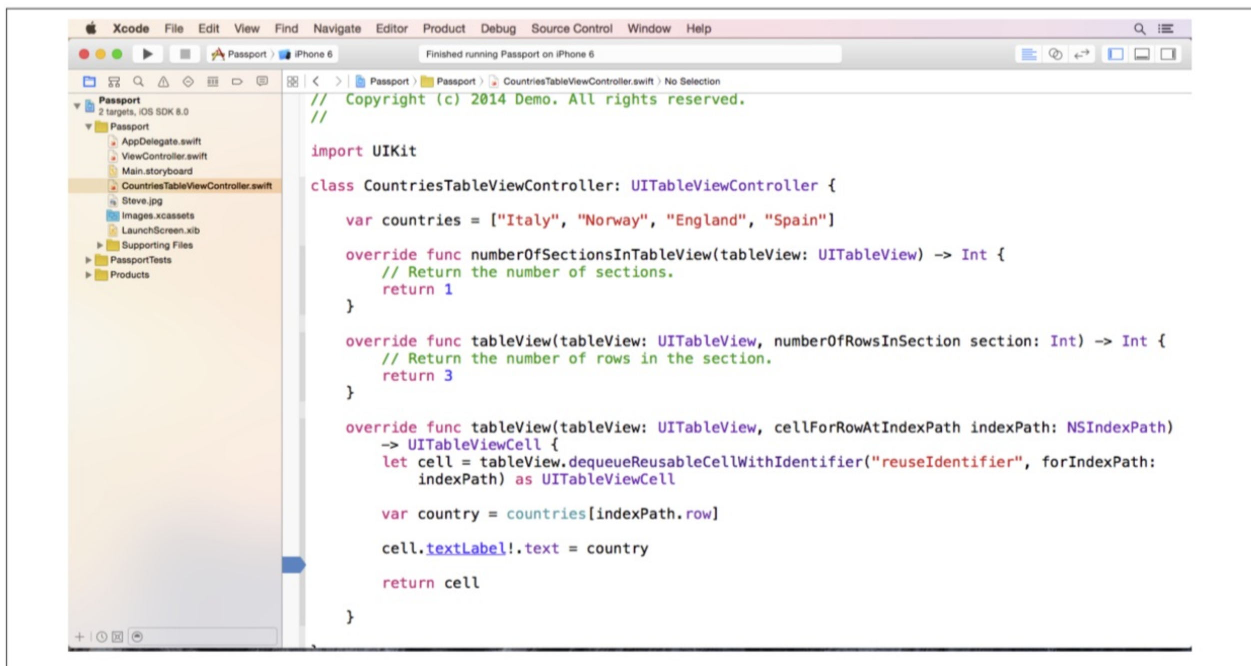


Page 178 | Chapter 6 : Next Steps : Debugging, Documentation, and App Icons

Spain并没有出现在table view中，关闭iOS模拟器，然后回到*CountriesTableViewController.swift*文件。

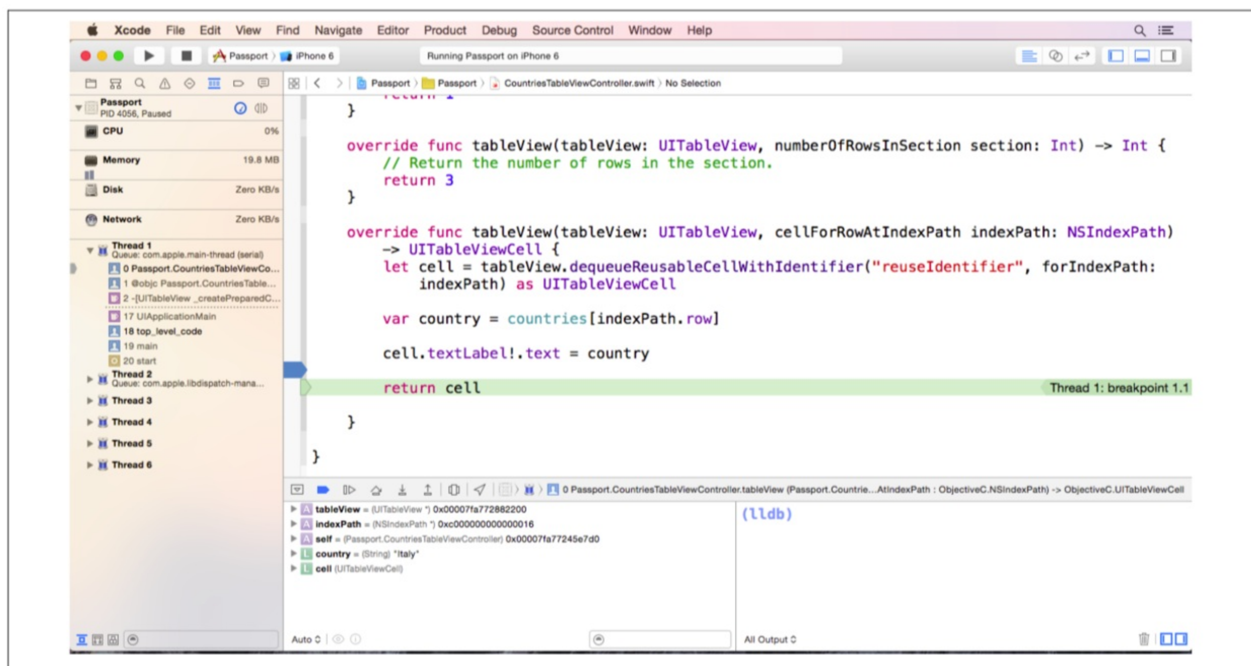
滑倒cellForRowAtIndexFath方法，把鼠标放到 `cell.textLabel.text = country` 下方，Editor左侧有一条灰色的区域，这块区域叫做gutter，点击 `cell.textLabel.text = country` 下方那行的gutter，生成一个breakpoint的（断点）。

生成breakpoint后会在gutter上出现一个蓝色的箭头，这个蓝色箭头就是breakpoint（图6-16）。breakpoint会暂停程序运行，允许你一步一步的检查每一行代码。用这种方法，可以能够找到问题所在。点击Play按钮（Command+R）。



Exercise: Expanding the Passport App | Page 179

App启动后，点击Show Countries按钮，这时模拟器会隐藏起来，Xcode出现在最前面，有breakpoint（断点）的那行代码会高亮绿色背景，Debugger也会出现在Xcode最下方（见图6-17）。



Page 180 | Chapter 6 : Next Steps : Debugging, Documentation, and App Icons

Debugger左侧的清单列出了所有的变量，这个清单叫做variables view。请注意，country变量目前是设置成了Italy。点击debug tool（调试工具栏）中的Continue按钮，从左往右数第三个按钮，外表看起来像是Play按钮。

下载的指示会立即出现，然后breakpoint所在的那行代码被标注出绿色背景，再看country这个变量，现在被设置成了Norway。然后继续点击debug tool（调试工具栏）中的Continue按钮。

下载的指示会立即出现，然后breakpoint所在的那行代码被标注出绿色背景，再看country这个变量，现在被设置成了England。然后继续点击debug tool（调试工具栏）中的Continue按钮。

这时模拟器再次出现了，table view中还是和上次同样的三个国家。调试区域显示cellForRowAtIndexFath方法被调用了三次。

打开*CountriesTableViewController.swift*文件，去掉gutter上的breakpoint（断点），看一下*numberOfRowsInSection*方法，返回值是3，这意味着table view中只有3个cell，把返回值那行代码修改为：

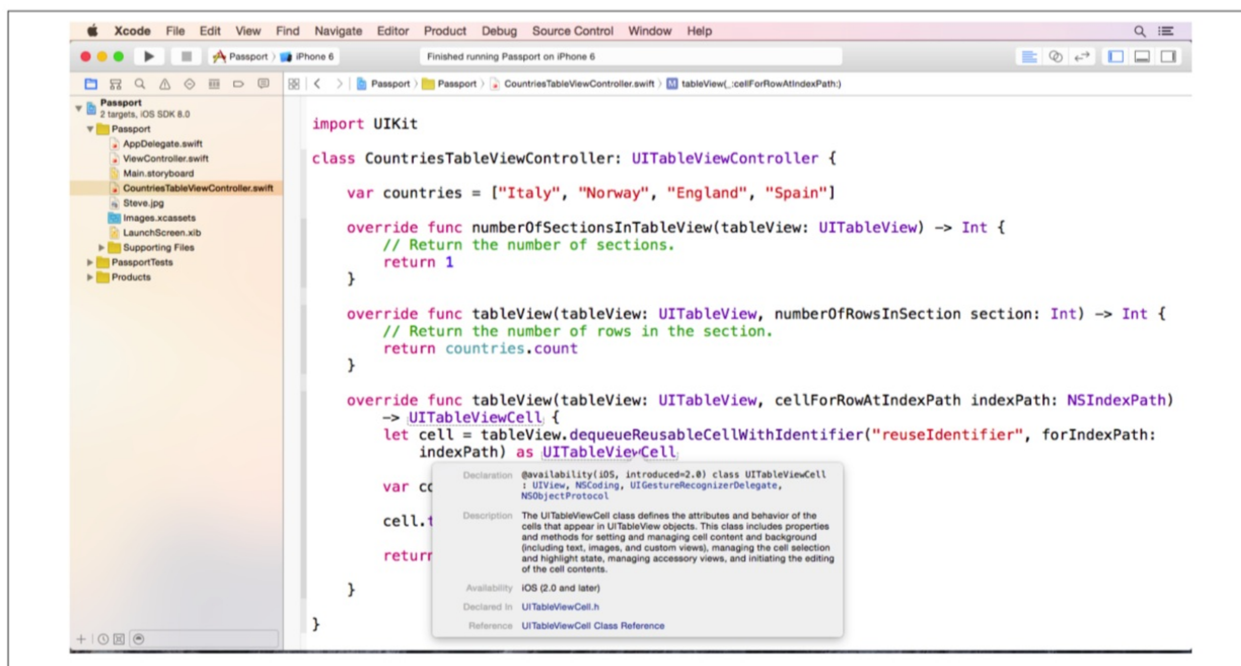
```
return countries.count
```

这行代码会让cell的数量等于数组countries中的元素数量。这样，countries数组中每个元素都会有一个cell，所有的元素现在都能够显示到table view中了。点击Play按钮（Command+R）。

恭喜你！你刚刚修复了你第一个bug！

Documentation文档

要是cell还能显示更多东西就好了。找到cellForRowAtIndexFath方法，按住Option键，点击方法中UITableViewCell这个词，会出现一个快速帮助框（见图6-18）。点击*UITableViewCellClass*链接。



Exercise: Expanding the Passport App | Page 181

出现UITableViewCell文档窗口，左侧是文档目录，往下滑找到accessoryTpye然后点击，文档会出现下面的内容：

The accessory view appears in the right side of the cell in the table view's normal (default)state. The standard accessory views include the disclosure chevron; for a description ofvalid accessoryType constants, see UITableViewCellAccessoryType.

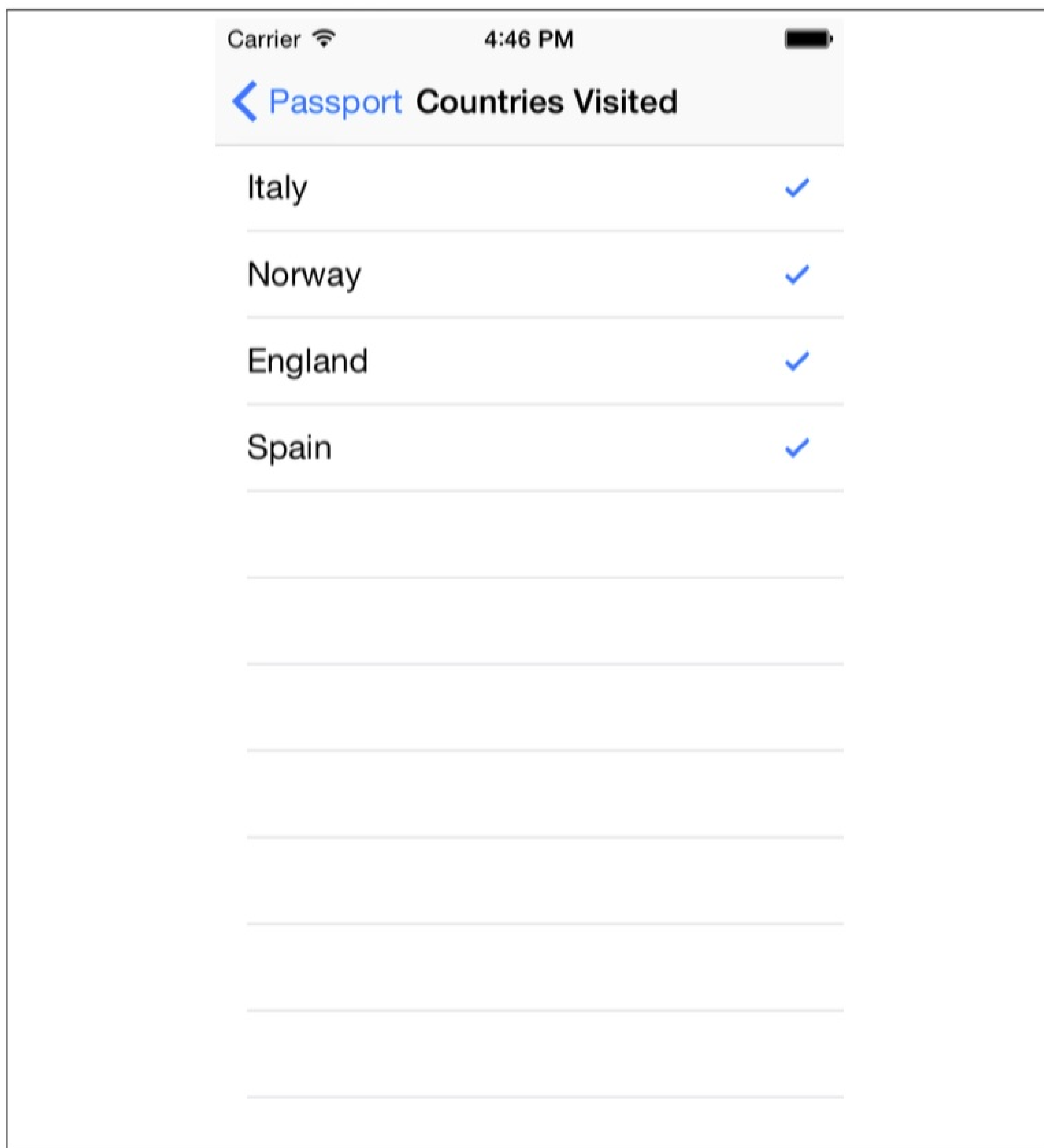
点击UITableViewCellAccessoryTpye，出现下列列表：

• None • DisclosureIndicator • DetailDisclosureButton • Checkmark • DetailButton

估计给每个国家增加一个check框会是不错的效果。关闭文档，把鼠标放到 `cell.textLabel.text = country` 下方，增加下方代码：

```
cell.accessoryType = .Checkmark
```

点击Play按钮（Command+R），现在tableview中每行都有checkmark了（见图6-19）。



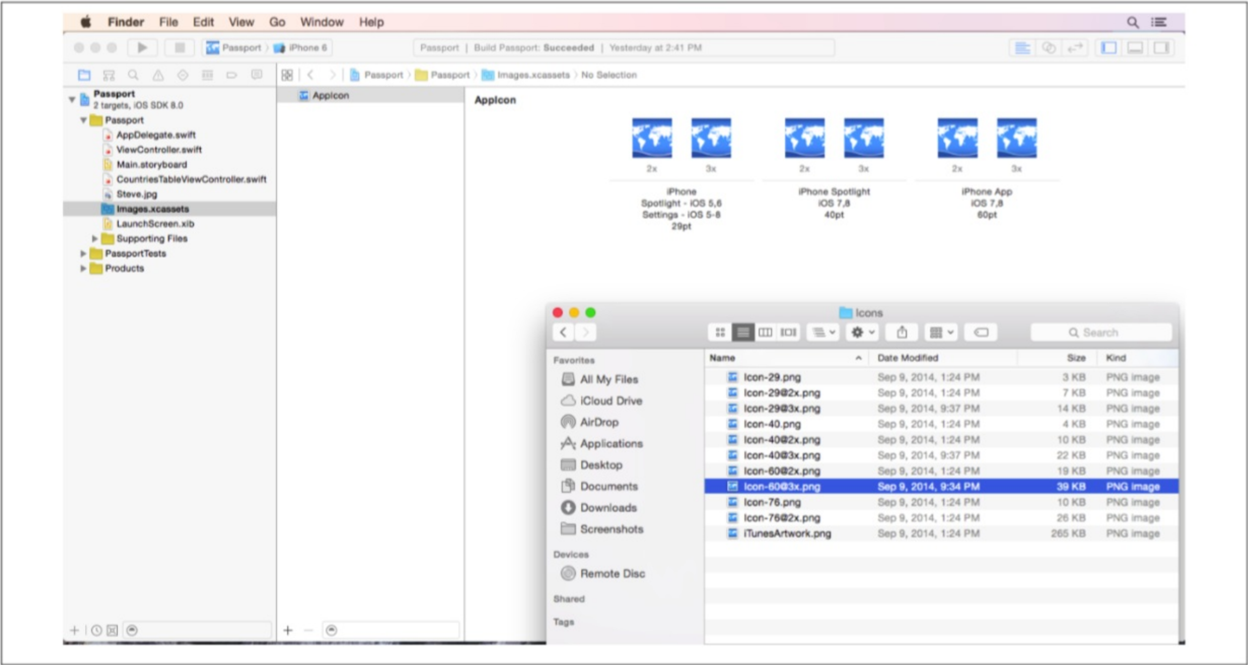
> Page 182 | Chapter 6 : Next Steps : Debugging, Documentation, and App Icons

App Icon应用图标

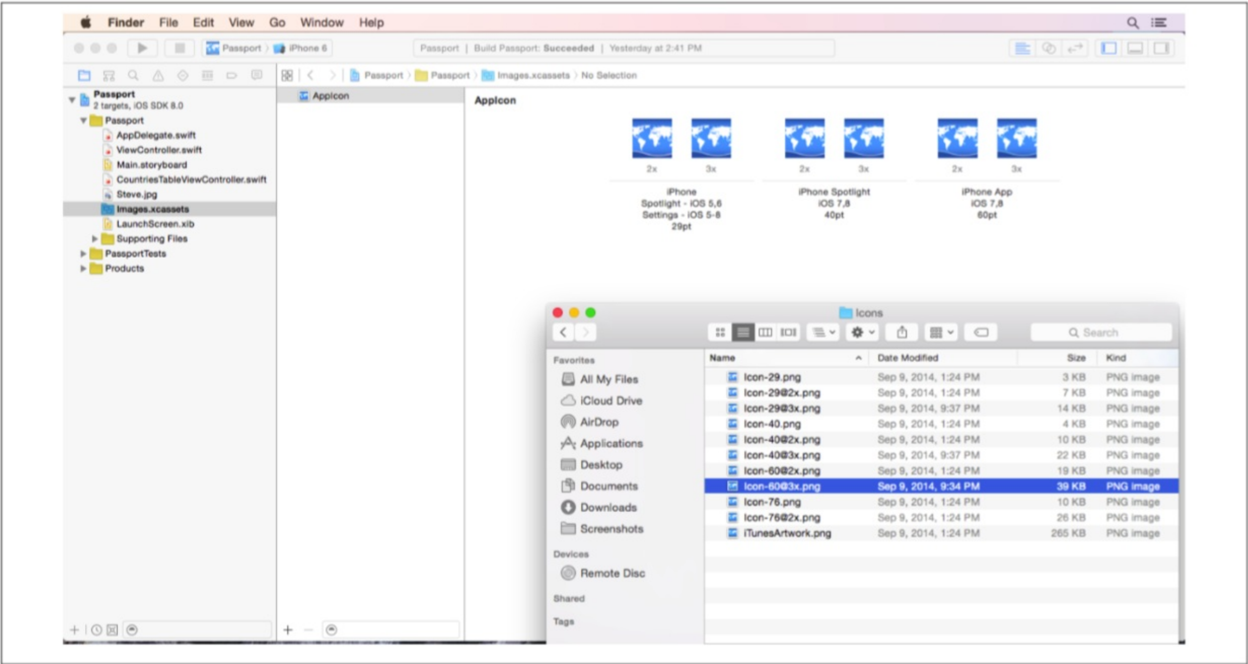
现在是时候给你的工程添加应用图标了。你可以从这个网站下载图标 http://www.AppSchool.com/book.（这个网站我是打不开，你们去网上下载免费的图标用吧）。

> Exercise: Expanding the Passport App | Page 183

接下来，打开文件夹，把文件夹放到图6-20这个位置，正好在Xcode的右下角，打开Xcode中的Images.xcassets，点击AppIcon，把图标拖到对应的位置。



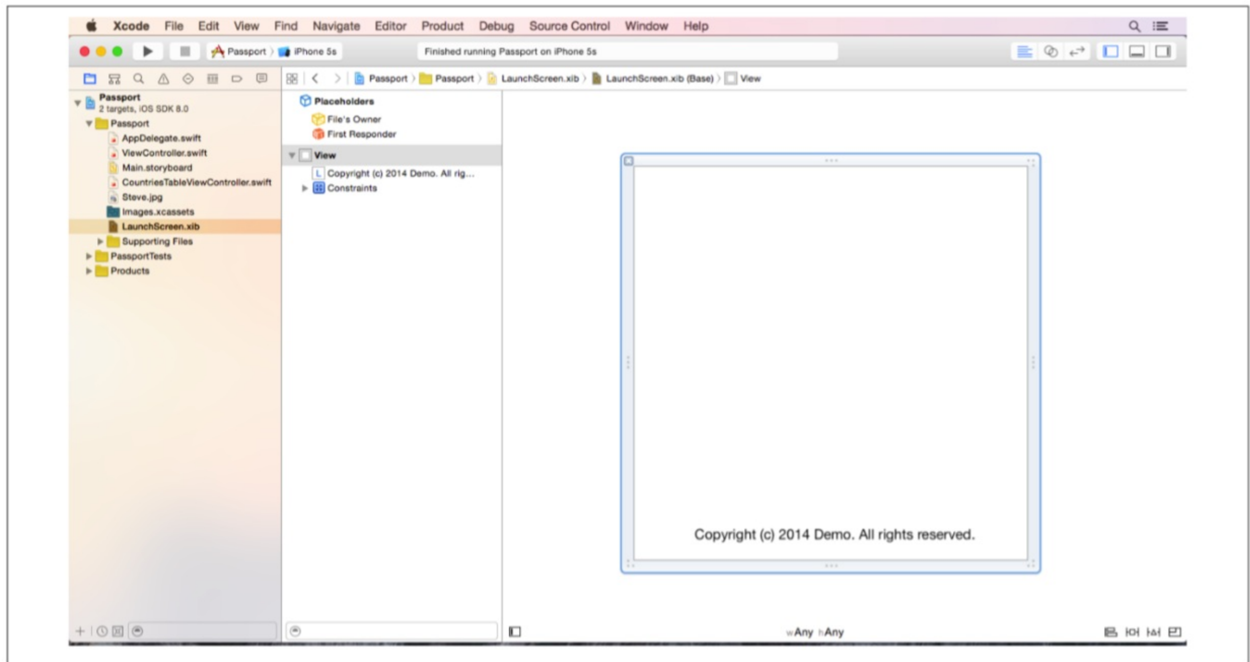
把每个文件放到对应的框中（图6-21）。



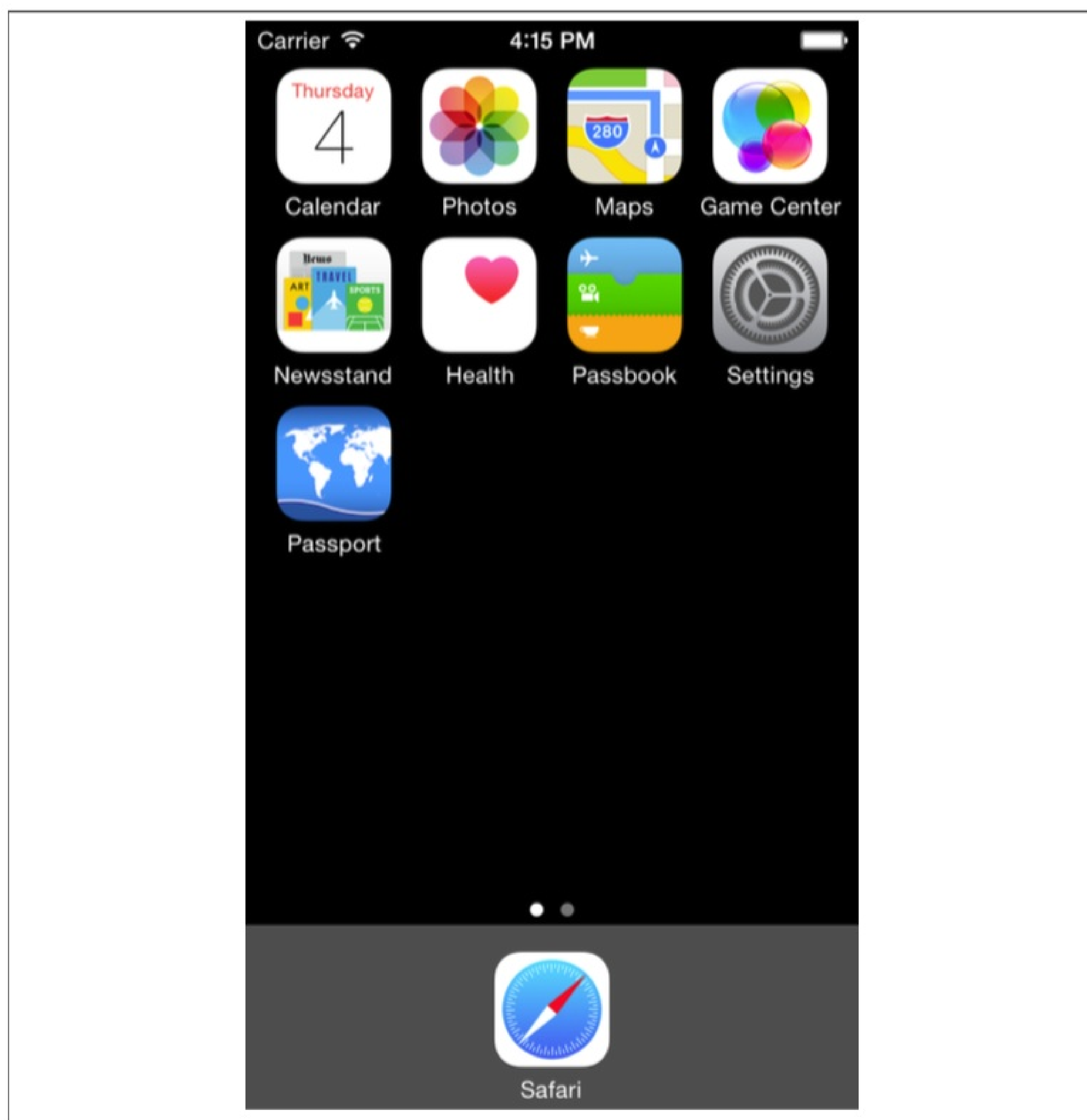
> Page 184 | Chapter 6 : Next Steps : Debugging, Documentation, and App Icons

App图标添加完毕。

打开LaunchScreen.xib文件，去掉Passport Label（见图6-22）。选中Label删掉。点击Play按钮（Command+R）启动模拟器。



点击模拟器，这时顶部菜单选择Hardware -> Home，可以看到我们刚刚设置的App Icon效果（图6-23）。



> Exercise: Expanding the Passport App | Page 185

如果App没有按照你想要的结果运行，或者程序有了错误或警告，不要太担心，学习的最佳方式就是试错，熟能生巧，到我们的网站上下载示例代码，对比一下代码，多试几次，直到搞定这个程序为止。

> Page 186 | Chapter 6 : Next Steps : Debugging, Documentation, and App Icons

第七章：设备和自动布局

在这一章节中，你将会学习如何为不同尺寸设备设计界面，我们会介绍所有的苹果设备及其尺寸。在这一章中你将学会设计的基本知识，从而可以为应用设计界面。

Screen Sizes（屏幕尺寸）

iOS系统是可以运行在多种设备多种格式下运行，包括iPhone，iPod Touch，iPad和iPad Mini。表格7-1包含了所有的设备及其像素尺寸。

表7-1 设备尺寸

Device	Height (px)	Width (px)
iPhone	480	320
iPhone 4 (Retina)	960	640
iPhone 5 (Retina)	1136	640
iPhone 6 (Retina)	1334	750
iPhone 6 Plus (Retina HD)	1920	1080
iPad	1024	768
iPad w/ Retina	2048	1536
iPad Mini	1024	768
iPad Mini w/ Retina	2048	1536

Retina Displays（视网膜屏）

Pixels（像素）表示显示器上可以绘制图片的基本单位，就像是方格纸上的一个小格子。随着设备的不断演进，屏幕上的像素值越来越多。

> Page 187

这就意味着屏幕的像素密度越来越高，每英寸可呈现的图像细节增加。Pixel-per-inch（PPI）像素每英寸为单位来表示影像分辨率的大小。当Steve Jobs发布iPhone 4时，他说人眼是无法注意到300PPI以上的区别，iPhone 4 有326PPI，接着苹果使用Retina来表示超过300PPI的设备，二iPhone 6 Plus有104PPI，用Retina HD表示。

为了帮你管理App上Retina和Retina HD图片，苹果公司提供了文件命名关键词方法：
Image.png 非Retina屏幕设备的标准尺寸图片 *Image@2x.png* Retina屏幕设备的2倍尺寸图片
Image@3x.png Retina HD屏幕设备的3倍尺寸图片

如果你在命名图片时，使用了 `@2x` 和 `@3x` 关键词，Xcode会自动为你匹配相应的Retina屏幕和Retina HD屏幕。

Swift需要在iOS7或者更高的系统下使用。iOS 7 只能在iPhone 4 以及更新设备上运行。因为除了iPhone 4，iPhone 4 以上的设备都使用了Retina屏幕，所以我们要保证适配Retina屏幕。有些iPad也在运行iOS 7，但是不是所有的运行iOS 7的iPad都有Retina屏幕，iPad Mini和iPad 2都可以运行iOS 7却不是Retina屏幕。所以1027*768这个尺寸，只有在适配非Retina屏幕iPad时才需要考虑。

Orientation（方向）

有许多设备运行iOS，每个设备的屏幕都有四个方向，一个设备同一时间可以显示一个方向：
*Portrait*垂直 这是默认的方向，设备通过这个定位可以让垂直方向的边比水平方向的边要长，Home键在设备的底部。 *Portrait Upside Down*纵向倒置 在这个定位下，设备的垂直方向边比水平方向边要长，Home键在设备的顶部。 *Landscape Left*左横屏 在这个定位下，设备的水平方向的边比垂直方向的边要长，Home键在设备的左侧。

> Page 188 | Chapter7 : Devices and Auto Layout

*Landscape Right*右横屏 在这个定位下，设备的水平方向的边要比垂直方向的边长，Home键在设备的右侧。

每个应用都要明确自己支持的方向。选定支持的方向后，应用要调整不同方向下的布局。默认情况下，支持Portrait,Landscape Left和Landscape Right三个方向。

Universal Apps（通用App）

App需要根据设备进行适配，苹果提供了三种格式帮助完成适配。

第一种格式是iPhone App。iPhone App将设计运行在iPhone和iPod Touch上。然而，iPhone App也可以运行在iPad上，因为iPad比iPhone有更多的像素，所以iPad要把iPhone上的App用缩放的形式展示，就是在iPad上出现一个iPhone尺寸大小的窗口，然后App就在这个窗口中运行，当然，也可以全屏展示，拉伸或者放大iPhone App，受到像素的影响，展示效果并不好。

第二个格式是iPad App。iPad App只能运行在iPad或者iPad Mini上，iPad App无法在iPhone或iPad Touch上运行，iPad App是专门针对iPad设计开发的，所以不会出现拉伸变形。

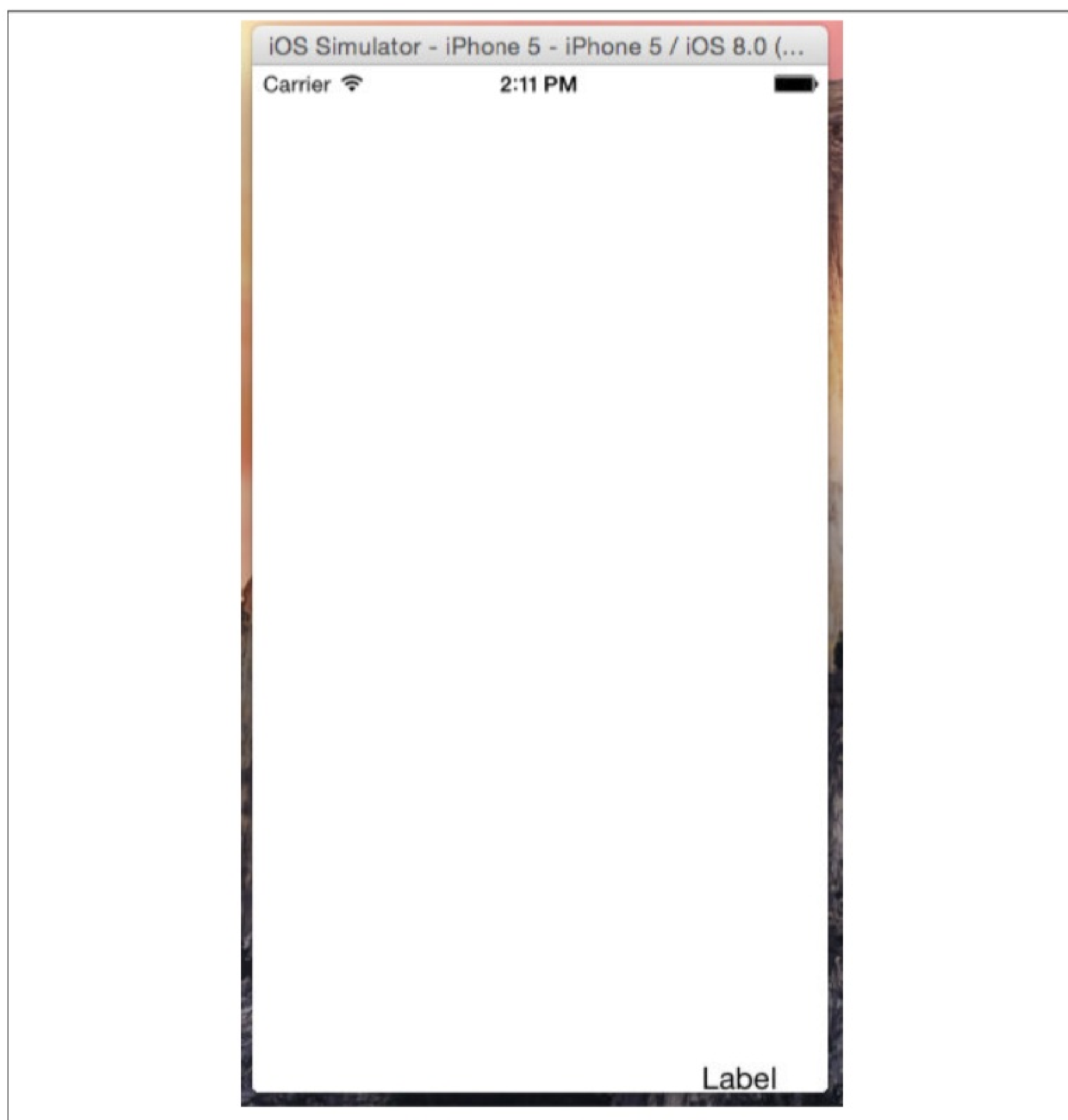
最后一个格式是通用型App（universal App）。universal app可以运行在任何尺寸下：iPhone，iPod Touch，iPad，iPad Mini。Universal app对iPhone和iPad各有一个具体的设计界面。为了帮助开发者让他们的设计效果在所有尺寸下都能实现，苹果提供了Auto Layout自动布局技术。

Auto Layout（自动布局）

自动布局可以给屏幕上的控件进行定位和控制尺寸大小。传统的系统使用固定的坐标和尺寸，但是Auto Layout使用规则（rules）来给控件定位。这些规则（rules）就叫做约束（constraints）。

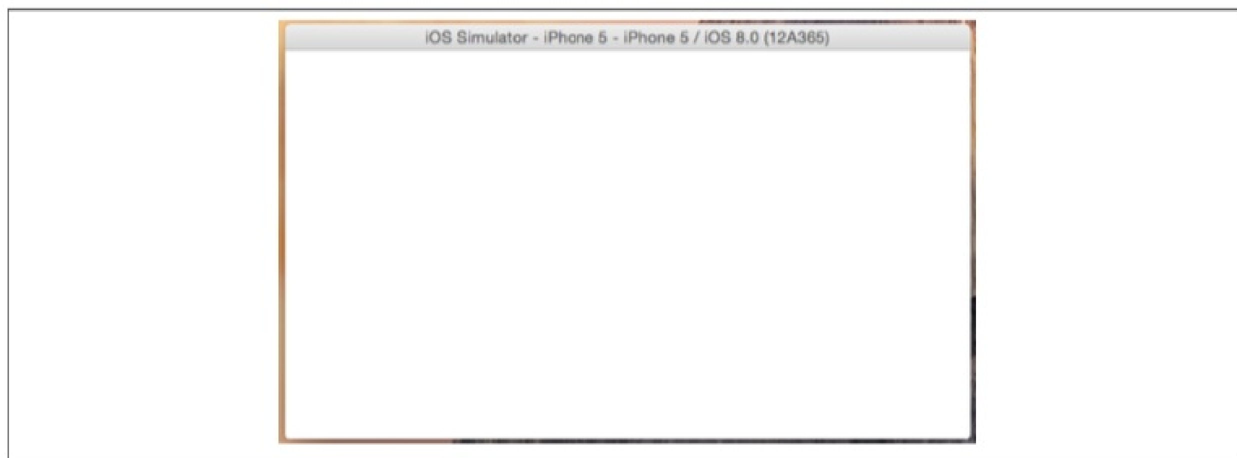
约束是有关尺寸和方位的规则，例如，Label的左边距离屏幕左边有20pts的偏移量。这种基于约束的系统可以让控件在不同尺寸不同显示器下自我调整。

想把一个Label放在iPhone 5的右下角，过去的方式是把Label设置成：Y-offset值550，X-offset值250，width值50，height值10（见图7-1）。



Universal Apps | Page 189

当需要考虑不同尺寸和不同方向下的布局时，这个方法就不行了。如果屏幕转向横屏，用这个方法设置的Label无法改变坐标值，无法出现在屏幕中了（见图7-2）。

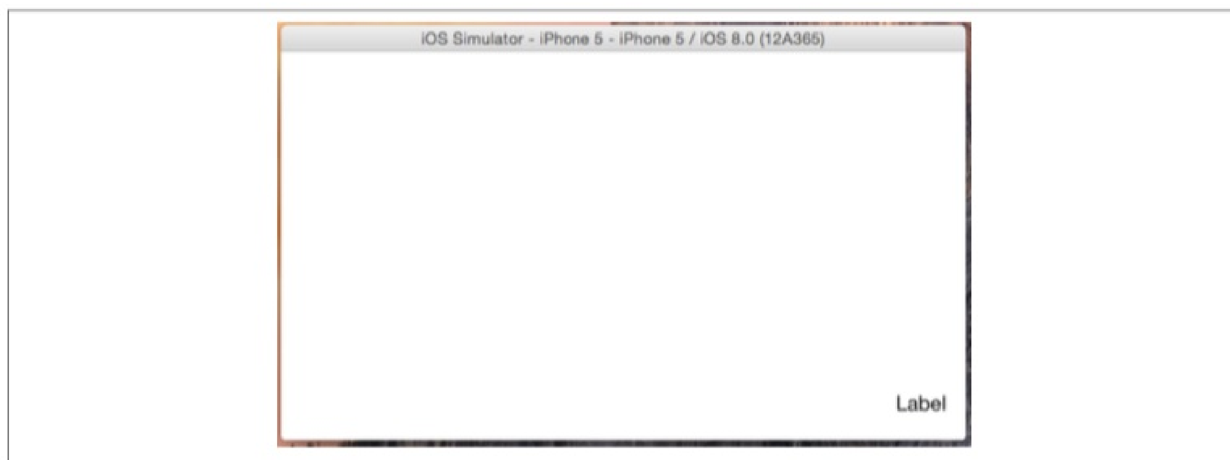


Page 190 | Chapter7 : Devices and Auto Layout

如果是使用Auto Layout来把Label放到右下角，Label不管是在横屏还是竖屏都会出现（见图7-3）。

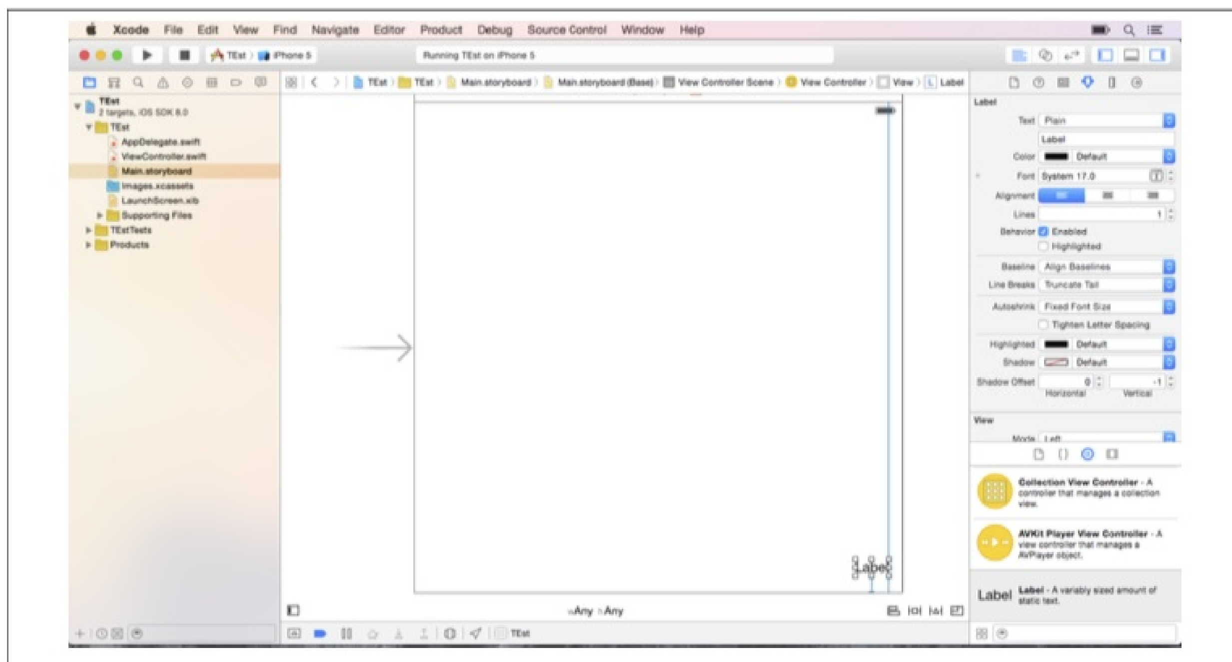
Auto Layout | Page 191

由于Auto Layout使用约束规则来约束控件，所以到设备转向横屏时，Label就会自动调整定位（见图7-4）。



Attributes（属性）

我们可以根据大量不同的属性来创建约束。例如，你把Label的右边距设置为距离containning view边缘30pts（见图7-5）。containing view就是把控件封装在一起的view（视图）。除了可以定位到某个控件右边，还可以定位一个控件的很多属性。



> Page 192 | Chapter7 : Devices and Auto Layout

left（左）、right（右）、top（上）和bottom（下）边距是比较常见的定位属性。例如，英语是从左向右读写的，然而如果你想展示的语言不是从左向右读写的，比如阿拉伯语，那相应的间隔就需要翻转过来。除了左边距和右边距，苹果公司还提供了leading和trailing属性。在从左向右读写的语言中，leading和trailing功能和左边距右边距一样，在从右向左读写的语言中，左边距右边距的概念可以用leading和trailing属性。

我们还可以基于width（宽）、高（height）、centerX（水平居中）和centerY（垂直居中）来创建约束。宽和高这两个属性一般是用来固定或者让多个控件等比变化，比如你想在屏幕下方一行排列四个按钮，当屏幕变宽时，这四个按钮就会等比变宽。

cuter和centerY属性是用来定位一个控件垂直居中或者水平居中。最后baseline属性只出现在有文字的控件中，指的是一行文字的底部边界。

Values（值）

每个约束都必须设定一个值，这个值可以是具体的数值，也可以是一个范围，也可以是Standard Value（标准值）。例如，在iPhone上，Standard Value（标准值）可能是20pts，而在iPad上，Standard Value（标准值）可能是25pts。Standard Value（标准值）是由苹果公司规定的，会根据不同的设备和方向进行变化。约束也可以设定为一个具体的数值，例如，距离Label的leading edge30pts偏移量。Auto Layout也可以支持范围或者不等量的值。例如，把约束设置为距离leading edge大于等于20pts，随着视图变化leading偏移量也会变化。最后，Standard Value（标准值）是默认的距离。

Intrinsic Size（固定尺寸）

在某些情况下，我们很有必要允许控件能自己调整尺寸大小，例如，有一个固定宽度的Label，显示的一段英文文字“Tap here to enter”，当这段话翻译成德语的“Tippen Sie hier, um geben”，会比英文更长一些，如果Label设置成固定宽度，那么德语的这段话就会被截掉一节。

为了避免这种情形发生，在Label这个控件中常常出现，Auto Layout有了一个Intrinsic Content Size选项，这个选项允许控件基于其内容来自动调整大小。开启这个功能方法：在界面上选中Label控件，点击顶部菜单栏Editor，然后点击Size to Fit Content。

Priority（优先级）

在大多数情况下，一个view会有很多约束，屏幕上的控件会按照约束进行自我调整。在某些情况下，多个约束彼此矛盾，优先级顺序决定了我们先使用哪个约束。

> Auto Layout | Page 193

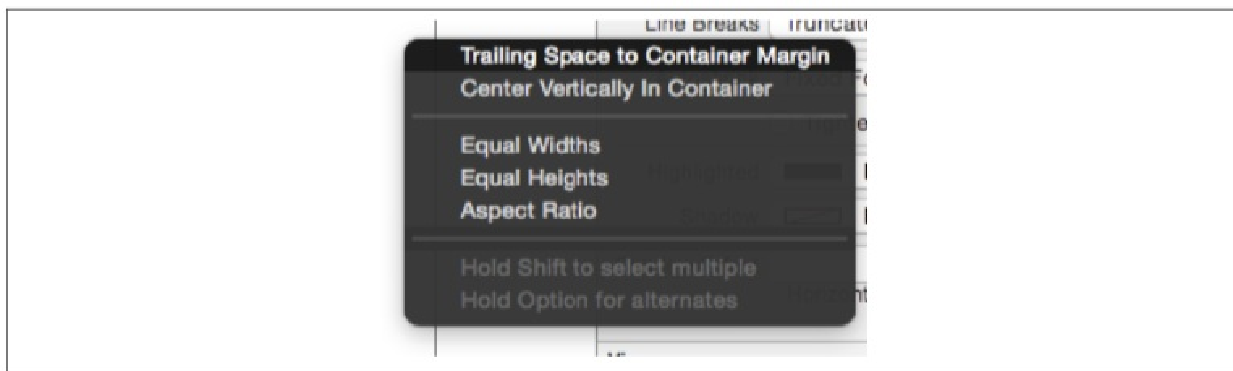
Creating Constraints（创建约束）

我们是在Storyboard文件中创建约束。当我们把控件拖到界面上时，这个控件是没有任何约束的。如果在没有设定约束的情况下运行应用，Xcode就会按照这个控件的绝对位置来定位这个控件，这意味着即使控件里的内容属性变化了，这个控件也不会移动。所以为了让控件能够

根据情况自我调整，就需要创建约束。每个控件都至少有一个水平和一个垂直的约束。有很多办法来给你的控件创建约束，下面我们来一一介绍。

The Control-Drag Method（Control拖动法）

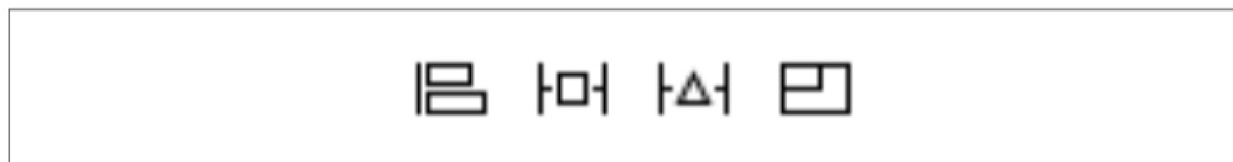
创建约束最方便的方法就是Control拖动法，选择一个控件，然后按住Control键，然后拖动鼠标拖向另外一个控件，接着弹出一个菜单窗口（见图7-6），根据你拖动的方向，弹出框中菜单选项内容也会不同。



如果你是向右拖动，就会出现一个Trailing Space Constraint选项。如果你是向左拖动，就会出现一个Leading Space Constraint选项。不管你向左还是向右拖动，都会出现垂直居中对齐（Center Vertically in Container）选项。而向上拖就会出现Top Space Constraint选项，向下拖就会出现Bottom Space Constraint选项。不管你向上还是向下拖动，都会出现水平居中对齐选项（Center Horizontally in Container）选项。

Auto Layout Buttons（自动布局按钮）

当然我们也可以通过Storyboard Editor编辑器右下角的Auto Layout菜单来创建约束（见图7-7）。Auto Layout菜单有四个按钮，用来给控件增加自动布局的约束。

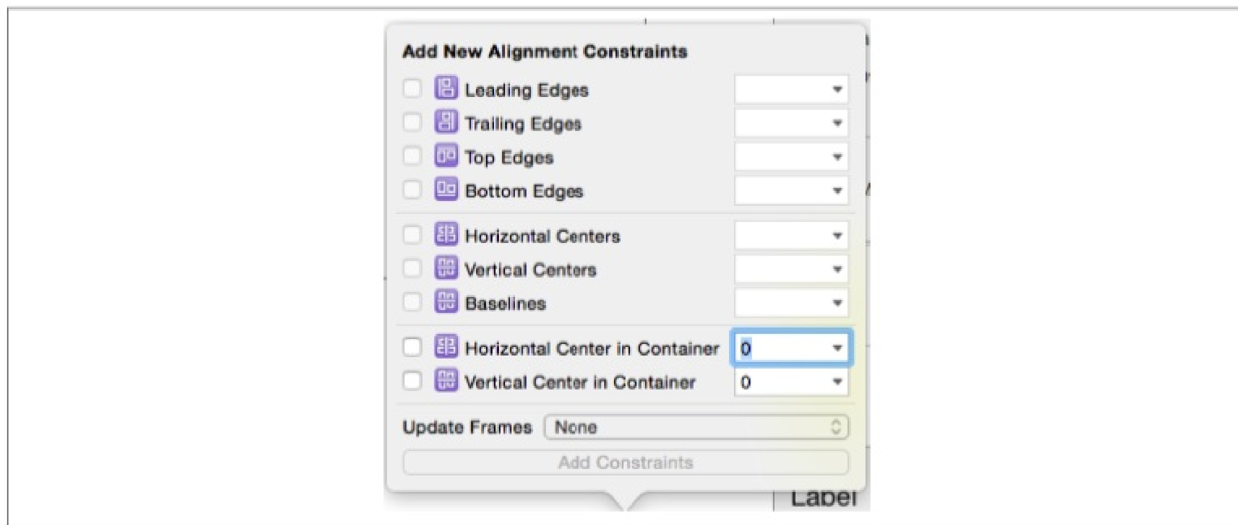


> Page 194 | Chapter7 : Devices and Auto Layout

The Align button

从左边起第一个按钮就是Align按钮，这个按钮是可以创建有关对齐要求的约束，例如让某个view居中，或者让多个相邻的view边对齐。

要使用这个按钮，首先先选定一个控件，点击最左边的这个Align按钮，然后弹出Align选项窗口，显示可能创建的约束（见图7-8）。勾选某个选项后在选项后面的输入框填写上偏移量，这个文本框同样也可提供一个下拉菜单，有Canvas（画布）和Standard Value标准数值2个选项。Canvas选项会根据你目前界面上控件当前的位置进行计算出偏移量，Standard Value是苹果公司根据最佳实践原则提供的尺寸，能够根据不同的设备进行变化。点击Add Constraints，就成功添加约束了。



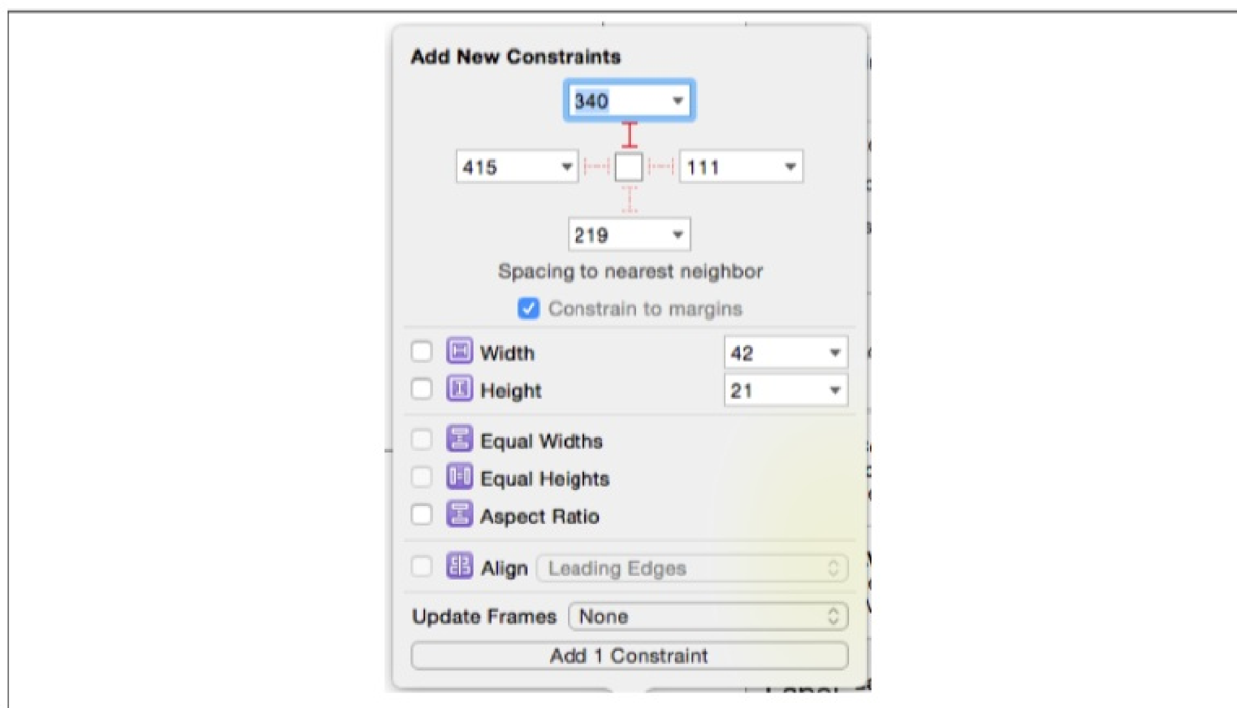
The Pin button

第二个按钮是Pin按钮。Pin按钮用来创建有关两个控件的距离或者控件宽高的约束。点击Pin按钮弹出Pin选项窗口，Pin选项从上到下有几个不同的部分。

所选控件距离上下左右的偏移量的约束，是由最上面的这部分部分来创建。

> Creating Constraints | Page 195

最上面这部分的输入框中输入数值，上下左右四个方向，每个方向都可以创建约束。这个地方有个棘手的问题值得注意，就是在创建约束的时候，你需要勾选输入框和小方块之间的红线，勾选后，红线会变粗（见图7-9）。勾选红线后，弹出框最下面的文案会更新为“Add 1 Constraint”。



从上往下第二部分是用来给控件设定固定的宽高约束，使用设定固定宽、高功能时要格外小心，尤其是在Label控件上，Label固定宽有时候会让Label的文字内容被意外截断。

从上往下第三部分是用来给多个控件设置等宽、等高、宽高纵横比，宽高纵横比约束功能是某个控件尺寸变化后，宽高比不会变化。

Align弹出框和Pin弹出框中都有Update Frames选项，点击Update Frames选项后，根据你更新的约束条件，让控件按照新的约束条件进行位置和尺寸的变化。你可以选择让所有的控件都按照新的条件进行调整，或者是仅限于让新的约束条件所影响的控件进行调整。

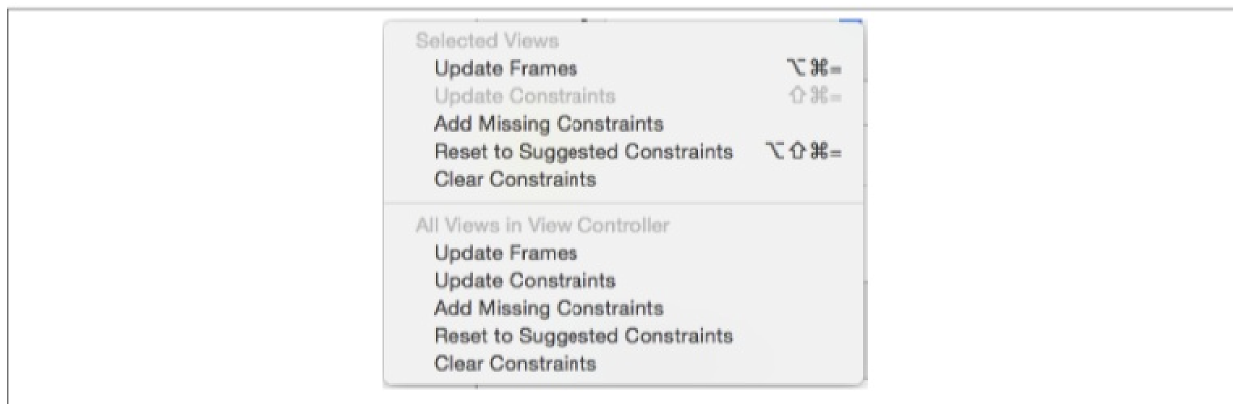
Pin选项中最后一个选项是Align选项，在创建约束时，这个选项最有帮助我们会经常用到。选中2个控件，Align选项会提供基于2个控件的对齐属性。想让一组左对齐的Label能够平均地放在一起，就可以使用这个选项。

> Page 196 | Chapter7 : Devices and Auto Layout

The Resolve Auto Layout Issues button

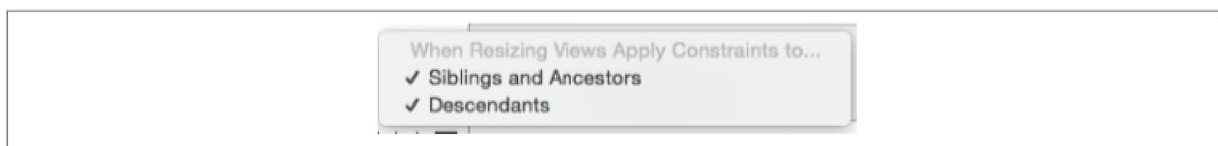
Auto Layout菜单从左往右第三个按钮是Resolve Auto Layout Issues按钮。这个按钮用来帮助我们修复Auto Layout中出现的问题（issues），它会提供修改建议，重新设置约束条件。还能基于当前界面中各个控件的当前位置进行分析生成创建约束。

Issues菜单分成上下两部分菜单（见图7-10）。上半部分的设置只影响到你所选中的控件，下半部分的设置会影响到所有的控件。Issue菜单中最有帮助的是“Reset to Suggested Constraints”选项。这个选项能够删除当前所有的约束然后基于当前控件的位置创建新的约束。这个选项是一个伟大的起点，允许约束在创建后可以被编辑。



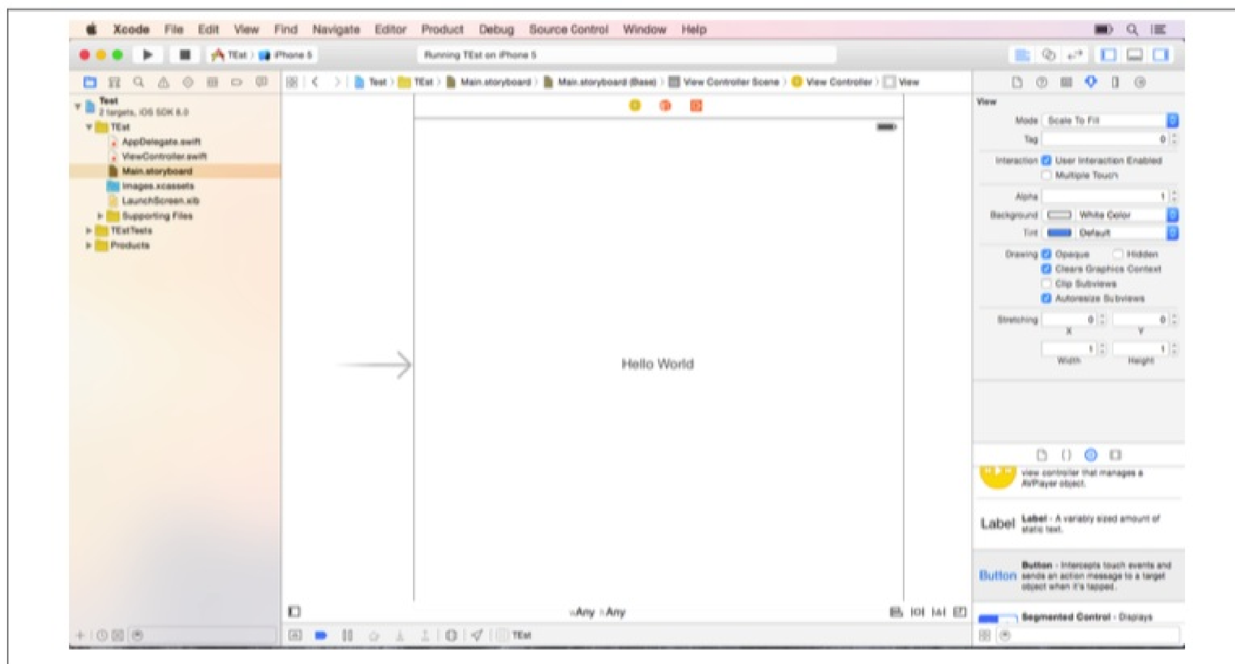
The Resizing Behavior button

最后一个按钮是Resizing Behavior按钮，点击这个按钮会弹出Resizing Behavior菜单，主要用来控制如何应用实现约束条件（见图7-11）。根据你选择控件的数量，有些功能是不能使用的。例如，equal widths选项能够让多个控件具有相同的宽，如果你选中了一个控件后，这个功能就不能使用。

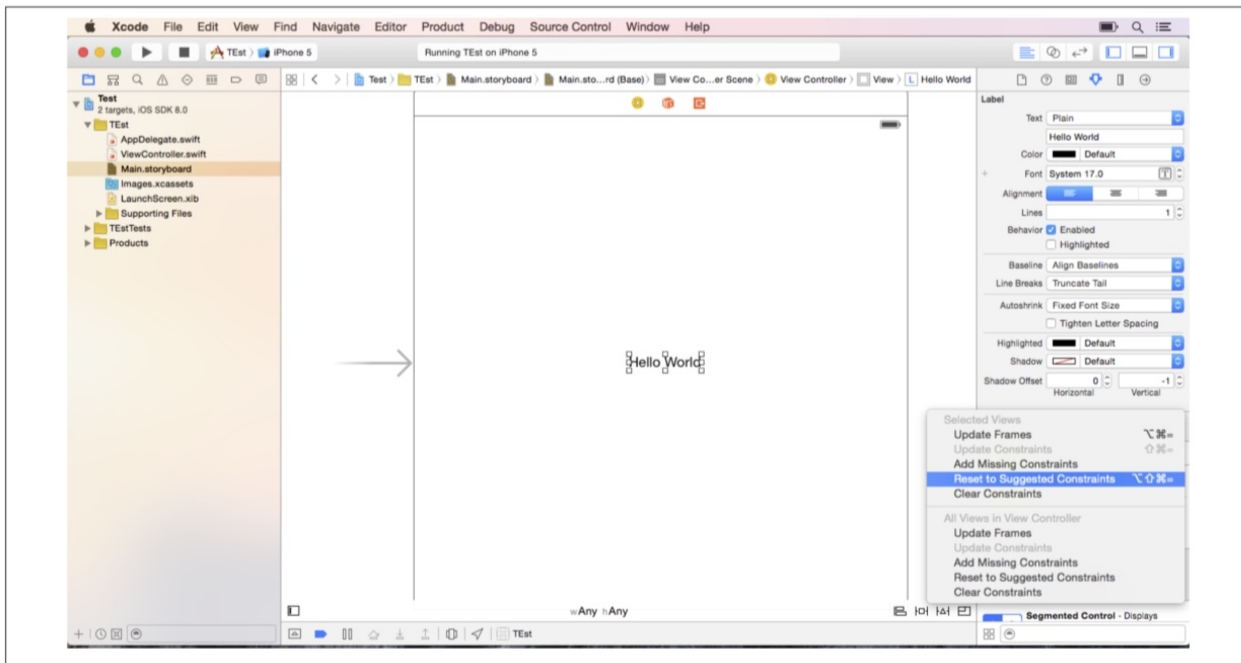


The Guidelines Method（辅助线方法）

当你在界面上来定位一个控件时，会出现垂直居中和水平居中两条辅助线（见图7-12）。Xcode提供的辅助线能够为你自动生成一些约束。



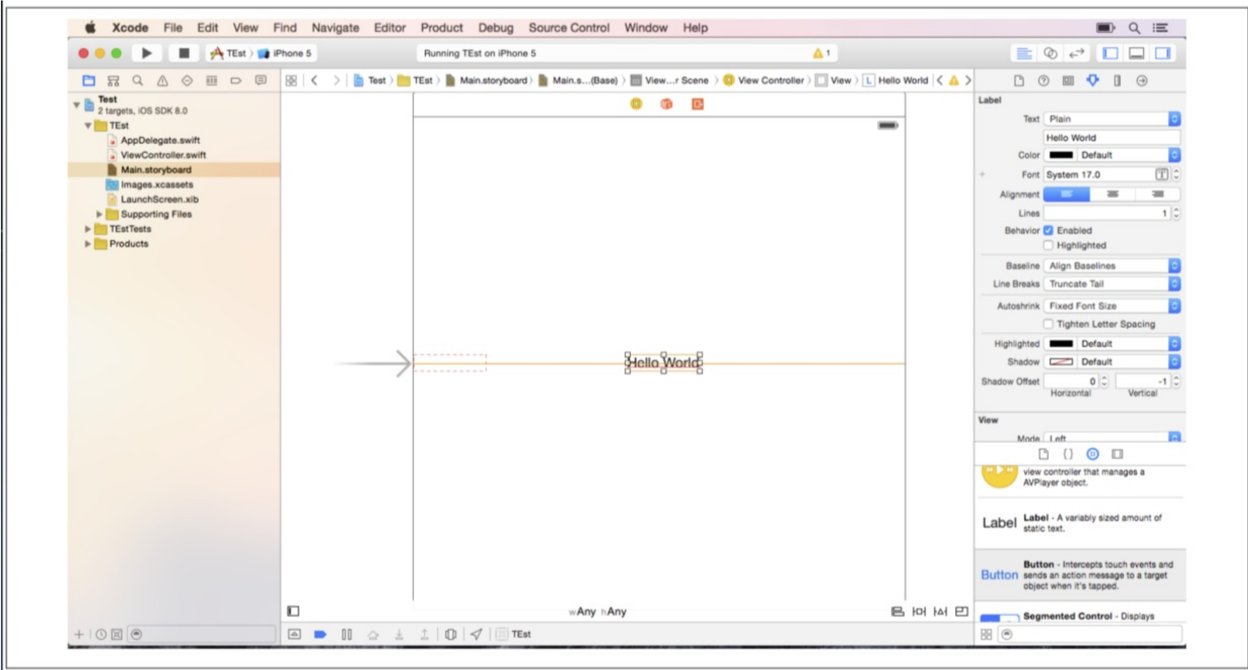
有线，我们要使用辅助线来定位一个控件，接着点击Resolve Auto Layout按钮（长得看起来像是两个翅膀之间有个三角），出现弹出框选项菜单，滑倒顶部选择“Reset to Sugested Constraints”（见图7-13）。这个选项会让Xcode删除目前的约束然后根据辅助线定义的位置自动创建新的约束。点开Document Outline，能看到所有的约束条件。



Page 198 | Chapter7 : Devices and Auto Layout

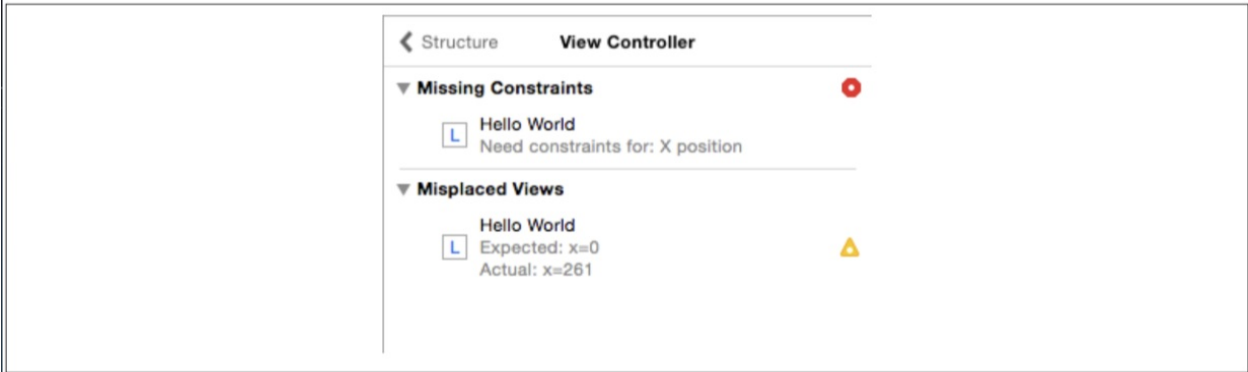
Testing Layout Constraints（检测Layout约束条件）

当你使用Auto Layout来定位一个控件，约束会通过颜色来提供反馈。如果约束的颜色是橘黄色的，表示目前的约束条件还不充分，需要更多的约束才能正确的定位这个控件，如果线是红色的，那么表示当前的约束是错误的（见图7-14）。

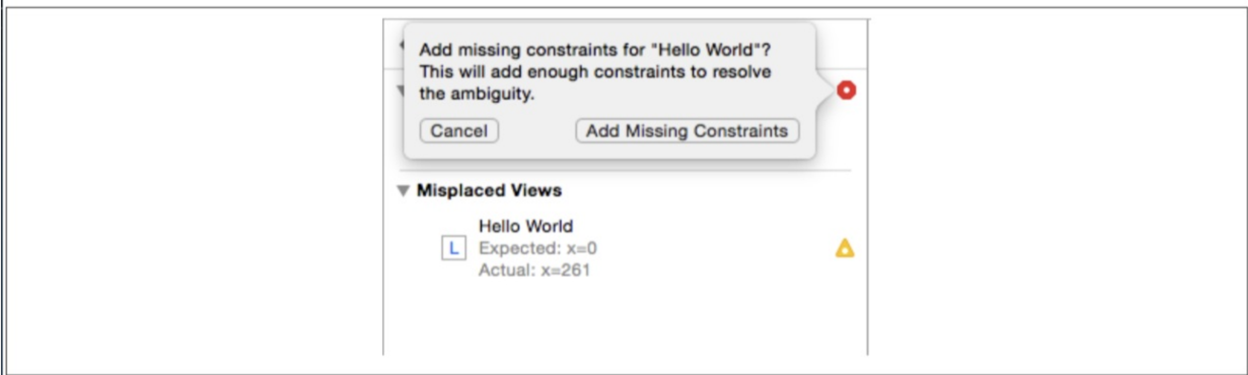


Testing Layout Constraints | Page 199

点击Document Outline，会出现黄色三角图标或者红色圆圈图标，这表示Auto Layout有了问题（Issue）。点击图标会出现问题清单，列出问题、原因（见图7-15）。

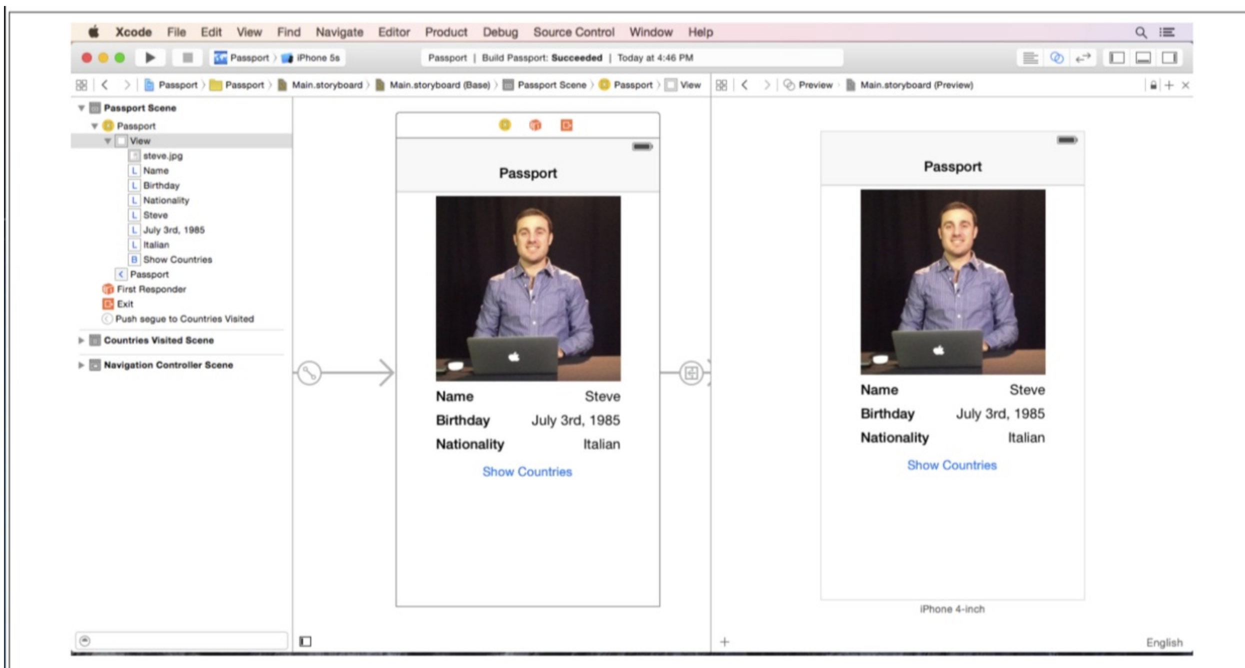


点击issue旁边的图标，会出现Fix-It弹出框（见图7-16）。弹出框展示建议的解决方法，点击Fix按钮会自动修复问题。一定要调查一下它给的建议，有时候它的建议并不是在所有的设备上能达到你想要的效果。



Previewing（预览）

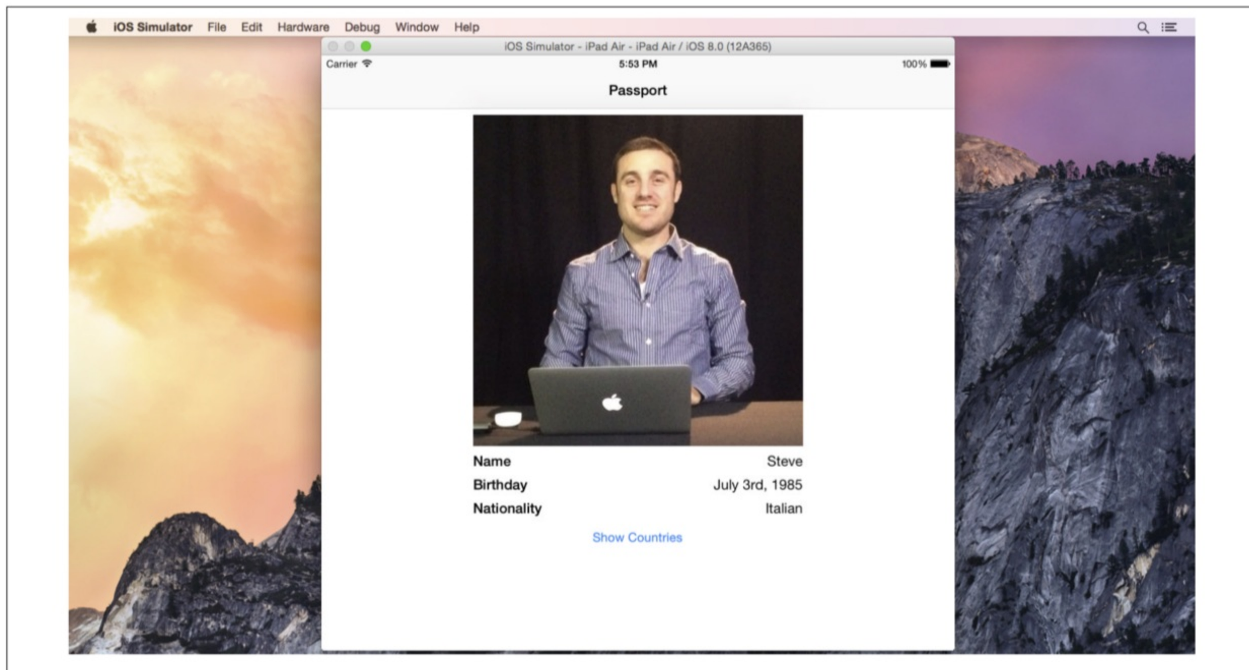
在多种屏幕尺寸来测试你的界面，这点很重要。为此Xcode提供了一个Previewer（预览窗口），展示在多个设备下具体的效果（见图7-17）。要打开Previewer，首先要隐藏Inspector，接着点击Assistant Editor，右边出现了一个窗口，点击这个窗口上方的Automatic按钮，下滑出一个窗口，点击Preview。右边的这个窗口展示当前界面在设备上的效果，点击左下角的加号，可以添加更多的设备。



Page 200 | Chapter7 : Devices and Auto Layout

第七章练习 Building More on the Passport App - 给护照App增加更多功能

本章练习我们继续使用第五章和第六章的练习Passport App，在本章练习中，Passport App会增加Auto Layout功能，App的界面会随着不同的设备（iPhone、iPad）不同方向（横屏、竖屏）进行自适应和自我调节（见图7-18）。



> Exercise: Building More on the Passport App | Page 201

打开Passport工程，点击Project Navigator中的Passport，显示工程详细信息。往下滑把Devices设置成Universal，接着勾选Upside Down、Landscape Left、Landscape Right。然后打开*Images.xcassets*文件，选中AppIcon，敲击键盘删除键。接着在白色的sidebar点击鼠标右键，选择New App Icon，一个全新的iPad和iPhone应用图标格子出现。然后打开图标文件夹，把对应的图标拖动到对应的格子中。

模拟器选择下拉菜单顶部工具栏Passport字样的右边。

下拉菜单中包括了所有当前模拟器可用的设备（见图7-19）。选择iPad Air，点击Play按钮。现在iOS模拟器启动iPad Air样式。

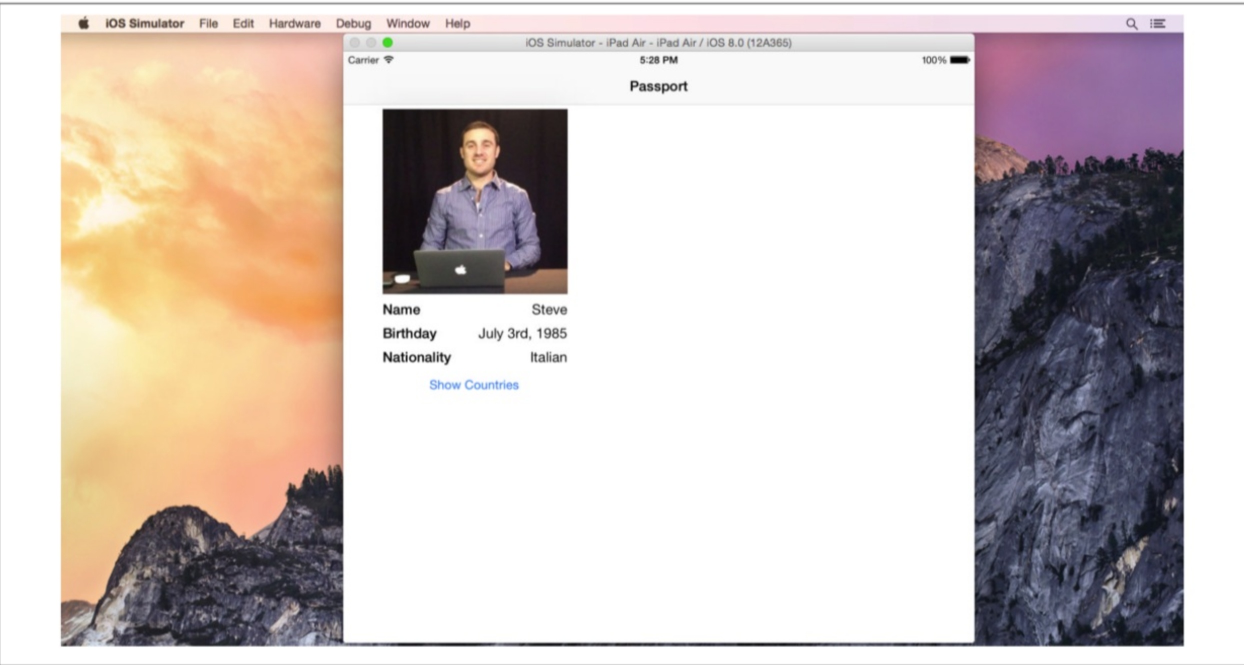


iPad Air的模拟器非常大，所以要调小一些，点击模拟器，然后顶部菜单选择Window -> Scale -> 50%（见图7-20）。iPad Air模拟器会调整成标准尺寸的50%。

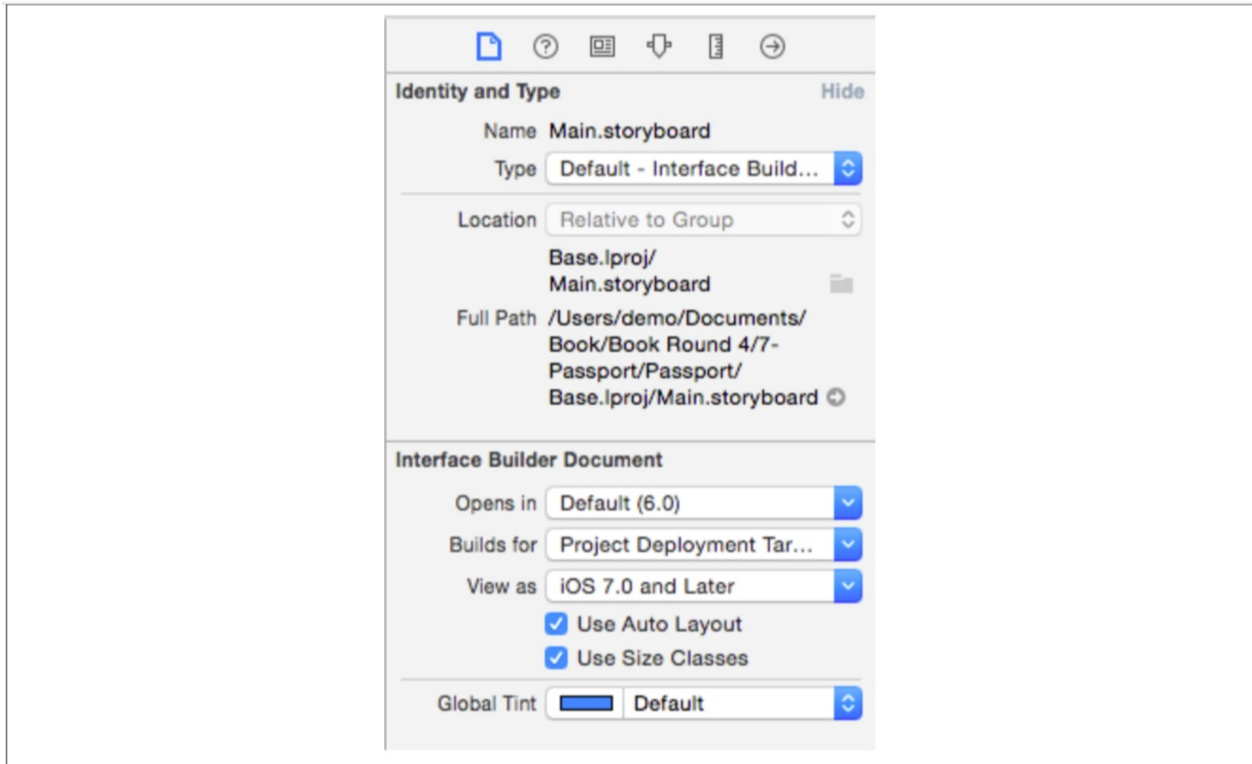


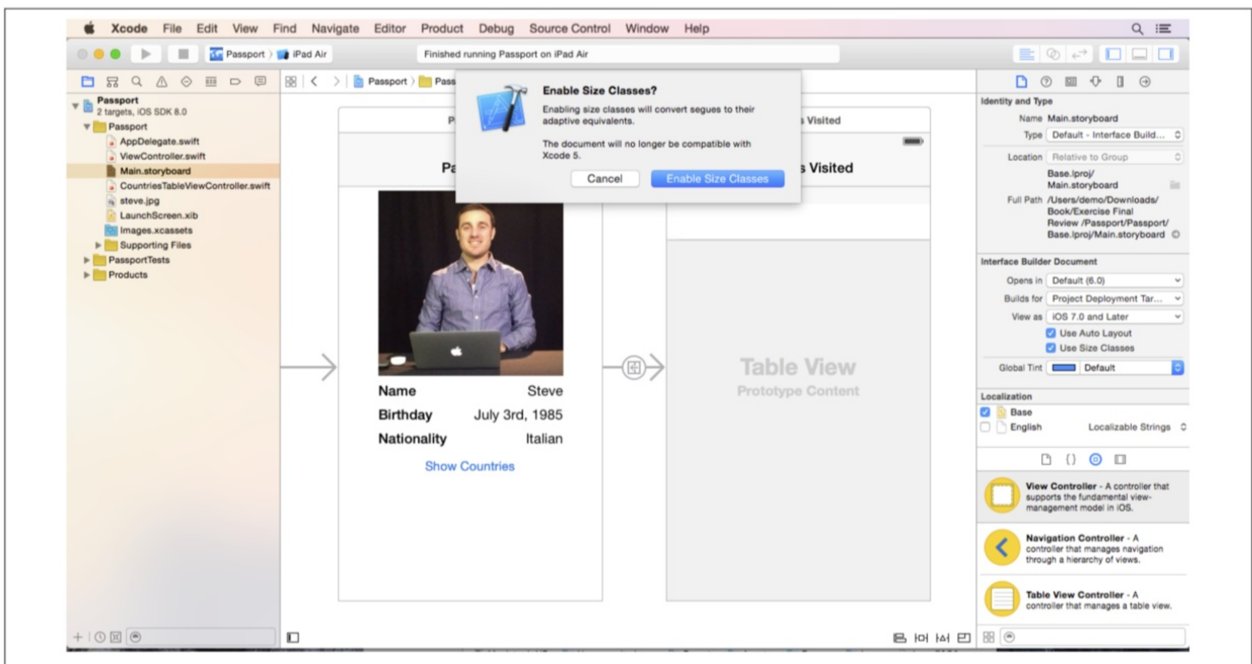
> Page 202 | Chapter 7: Devices and Auto Layout

Passport App在iPad上看起来并不好看。图片的位置变化了，然后labels也偏移了位置（见图7-21）。这是因为当前的控件位置使用的是固定的宽高坐标。



你需要把控件位置设置成Auto Layout。打开Main.storyboard文件，选择File Inspector（见图7-22），勾选Use Auto Layout，勾选Use Size Classes。点击Enable Size Classes（见图7-23）



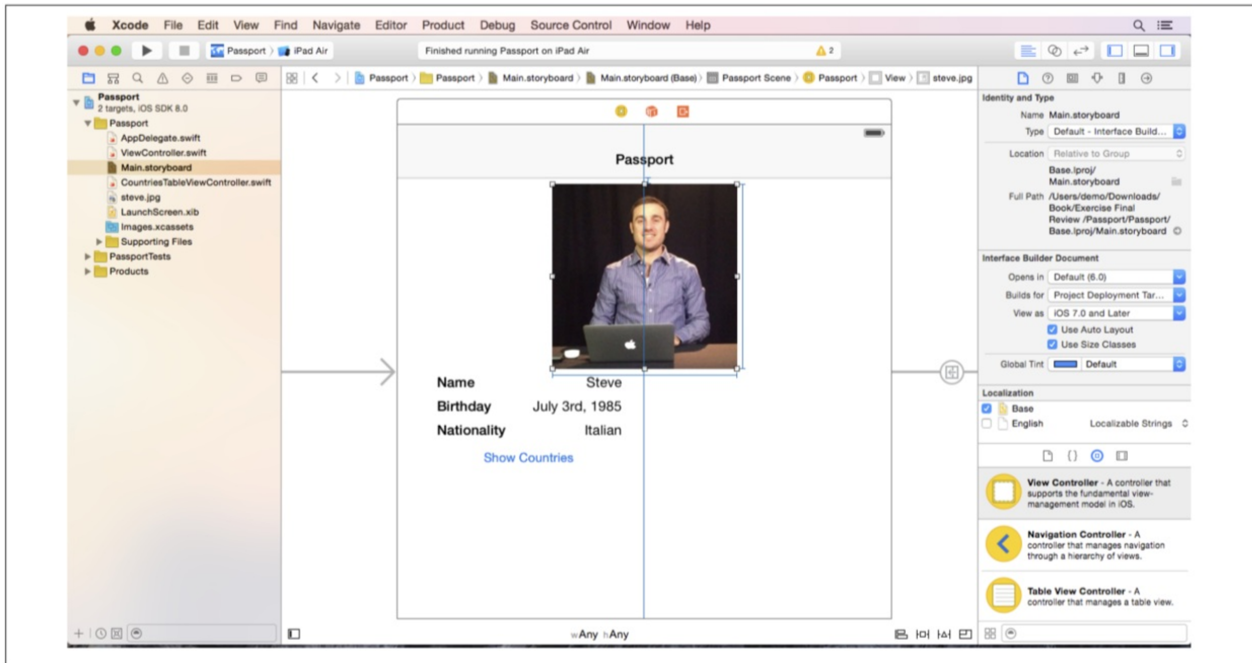


Exercise: Building More on the Passport App | Page 203

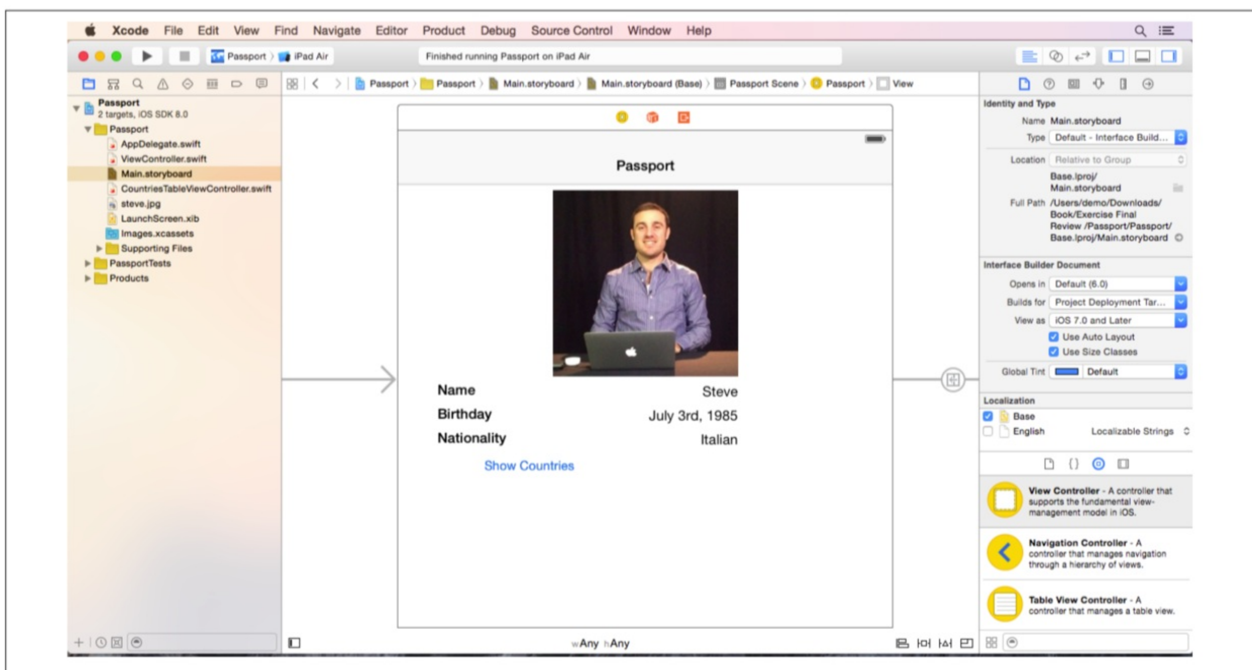
Size classes用来调整不同设备下的布局，现在我们可以使用Auto Layout了，要使用Auto Layout，首先我们要给控件增加约束（constraints）。打开Passport Scene。

Page 204 | Chapter 7: Devices and Auto Layout

借助辅助线把Image View放到界面的中间，然后把Image View往上移动直到上方出现一条水平的辅助线。点击Align按钮，选择Horizontal Center in Container，然后点击Add 1 Constraint。接着点击Pin按钮，上下左右四个方向选中上方竖线，值从下拉框中选择Use Standard Value，选中Width和Height两个选项，点击Add 3 Constraints（见图7-24）。

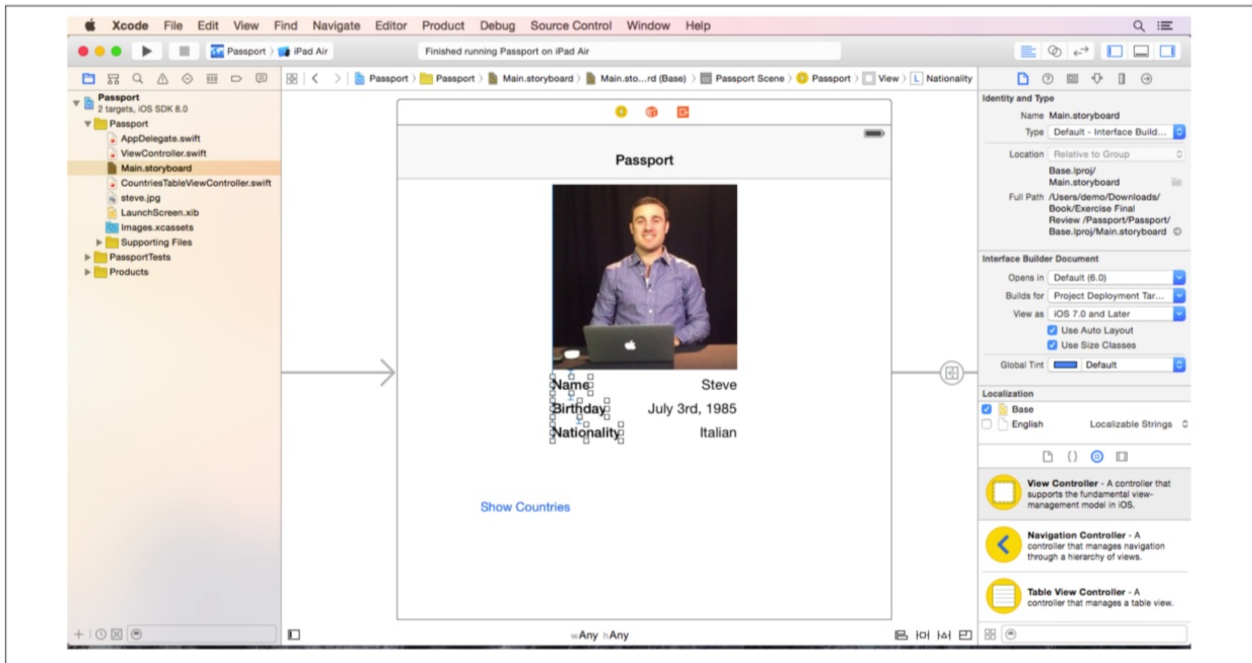


把右侧的三个Label放到Image View的右侧下方，Label的右边距和Image View的右边距相同对齐，选中这三个右侧Label，点击Align按钮，勾选Trailing Edges选项，点击Add Constraints。然后点击Pin按钮，勾选上方向竖线，选择Use Standard Value，Update Frames选项下拉选择Items of New Constraints，点击Add Constraints（见图7-25）。



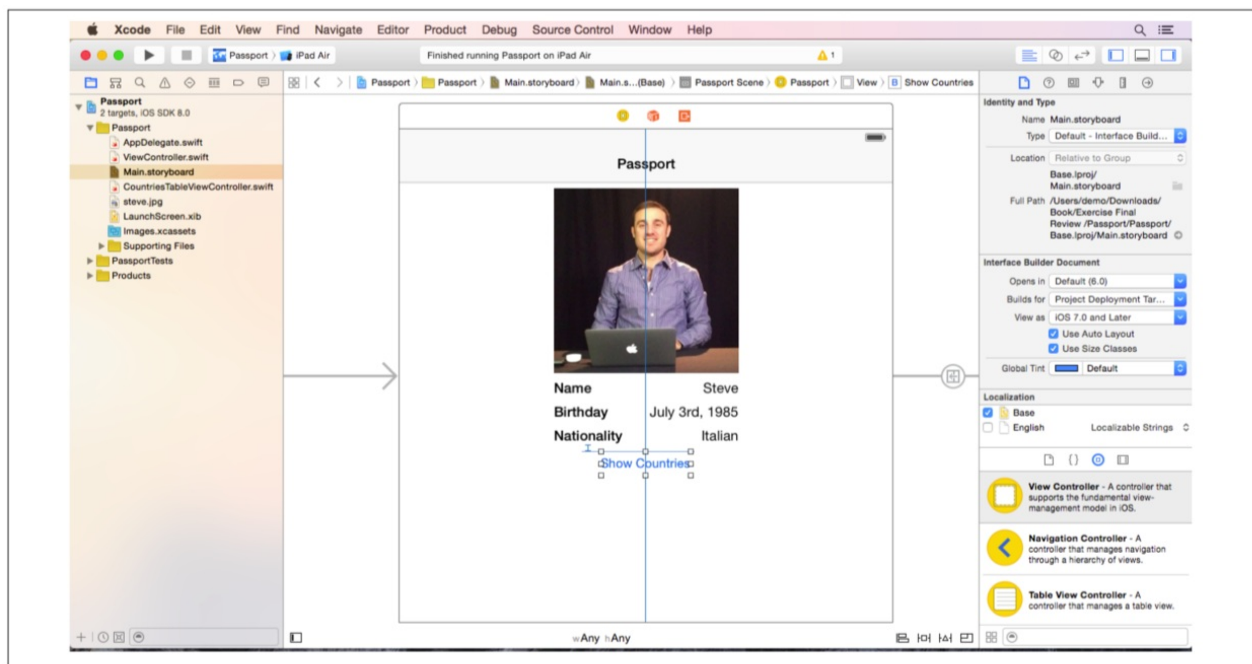
> Exercise: Building More on the Passport App | Page 205

把左侧的三个Label放到Image View的左侧下方，Label的左边距和Image View的左边距相同对齐，选中这三个左侧Label，点击Align按钮，勾选Leading Edges选项，点击Add Constraints。然后点击Pin按钮，勾选上方向竖线，选择Use Standard Value，Update Frames选项下拉选择Items of New Constraints，点击Add Constraints（见图7-26）。

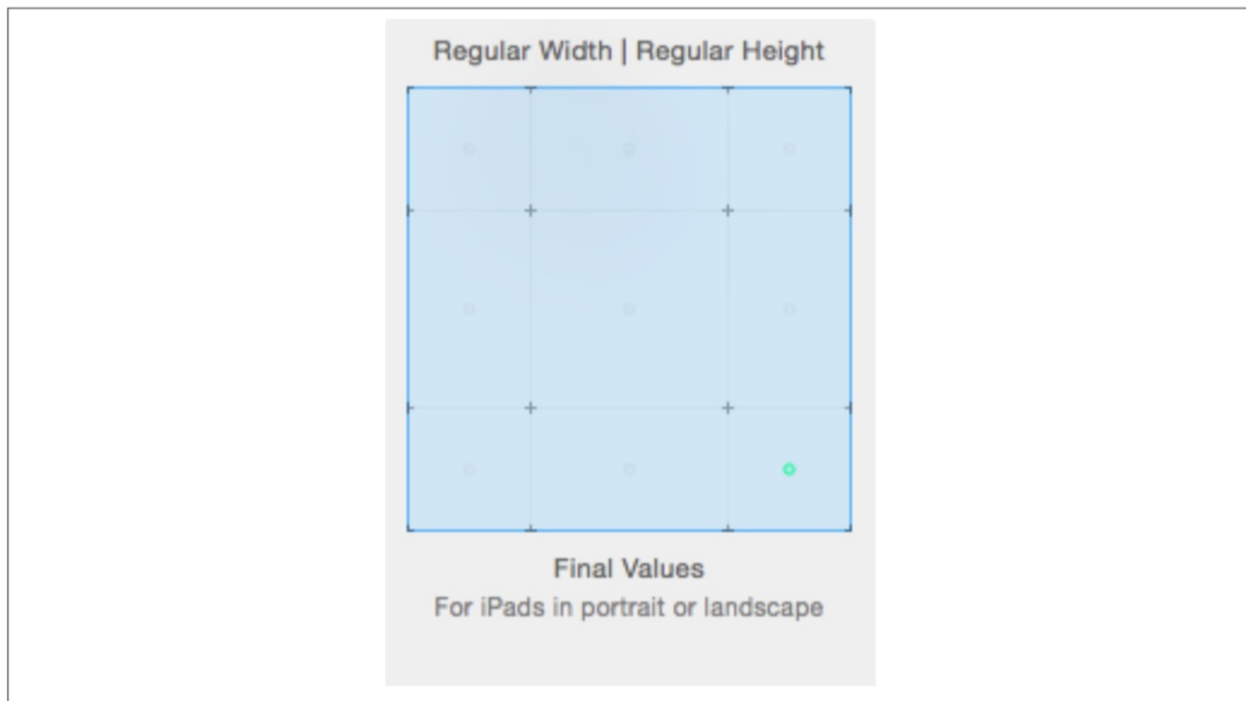


Page 206 | Chapter 7: Devices and Auto Layout

选中按钮，借助辅助线把按钮放到水平居中的位置，放到Label的下方。点击Align按钮，勾选Horizontal Center in Container，点击Add 1 Constraint，最后点击Pin按钮，勾选上方向竖线，选择Use Current Canvas Value，点击Add 1 Constraint（见图7-27）。

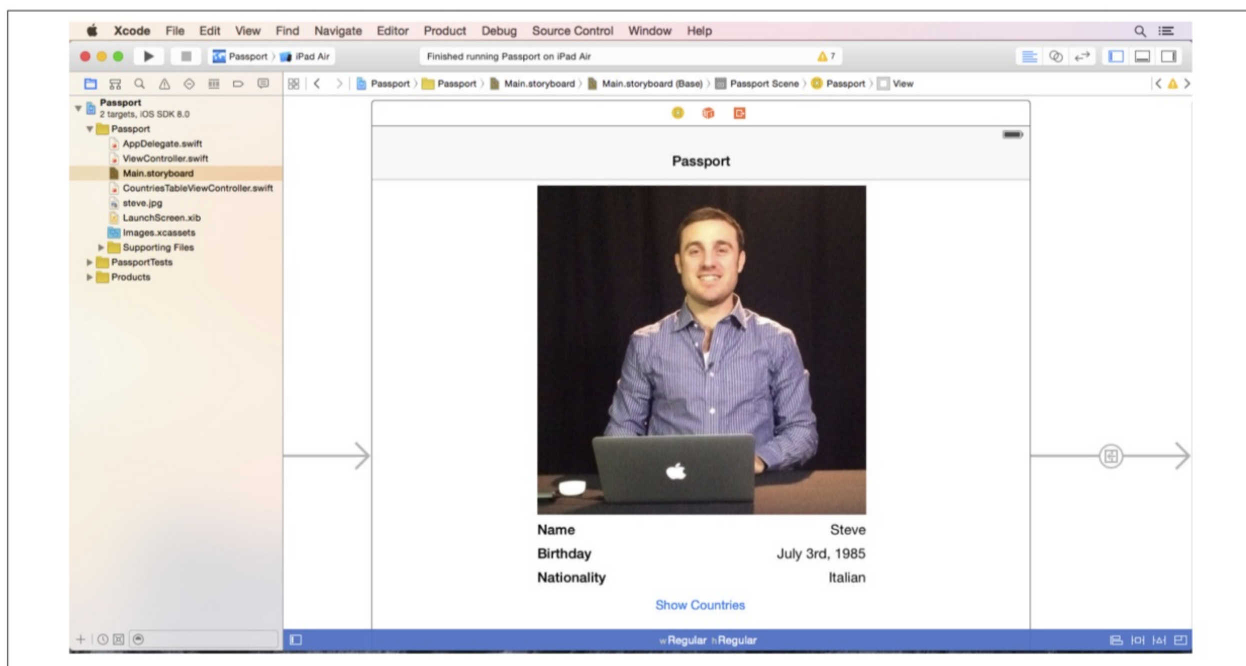


刚刚设置的这些约束会应用到所有的size classes和layouts中。现在，我们要为某个size class设置单独的约束了。点击最下方的“Any,Any”，会出现一个小的格子框，这里控制着当前我们编辑的是哪个size class，点击格子框的右下角（见图7-28），这个size class是用来设计iPad的横屏和竖屏。



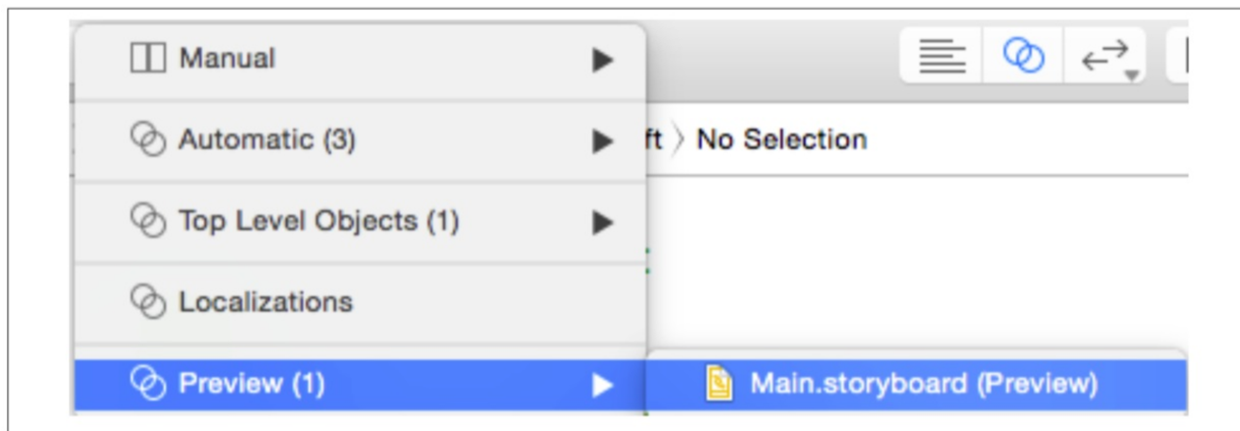
Exercise: Building More on the Passport App | Page 207

选中Image View，一条蓝色竖线出现在Image View的右侧，这时Image View的高度约束，双击这条约束，把值由225改为400，接着双击Image View下方的蓝色竖线，这是宽度约束，双击，把值由225改为400。Image View的大小已经重新设定了，Label和Button会自动调整它们的位置（见图7-29）。

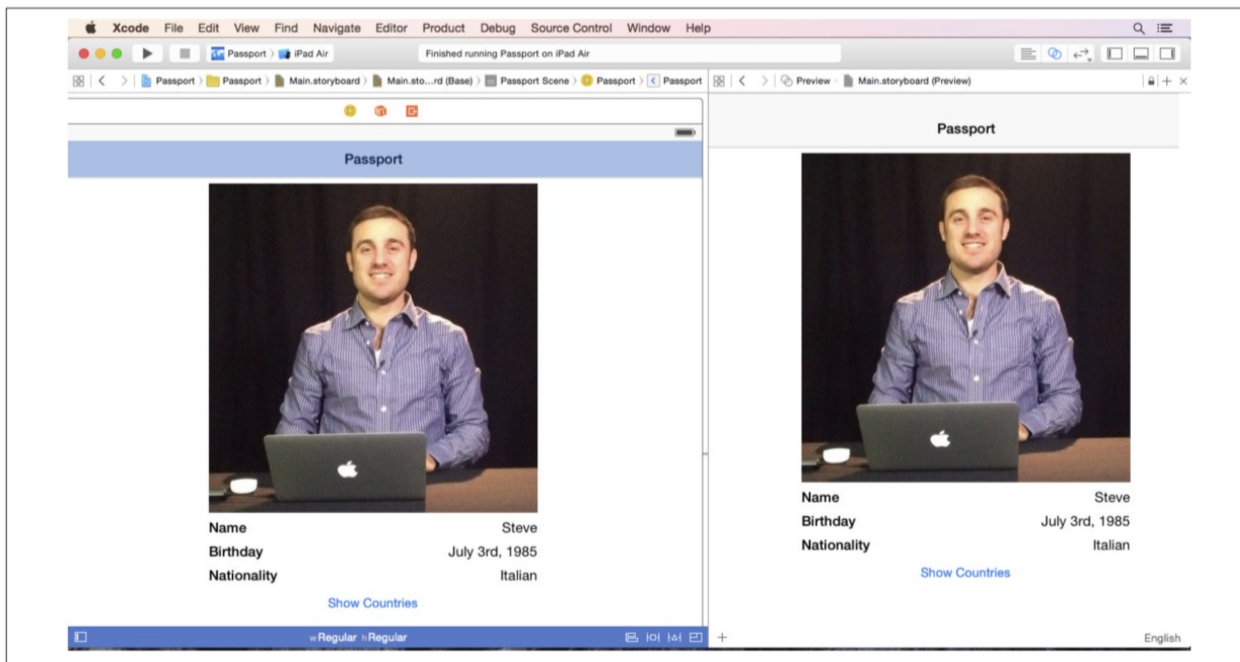


Page 208 | Chapter 7: Devices and Auto Layout

想要预览刚刚修改的约束效果，我们可以使用Previewer，在打开Previewer之前，我们最好给窗口腾出一些屏幕空间来，隐藏Project Navigator和Inspector，打开Assistant Editor，就是两个圆圈相交的图标，在Assistant Editor的工具栏中，点击Automatic，会出现一个下拉菜单，选择Preview -> Main.storyboard(Preview)（见图7-30）。



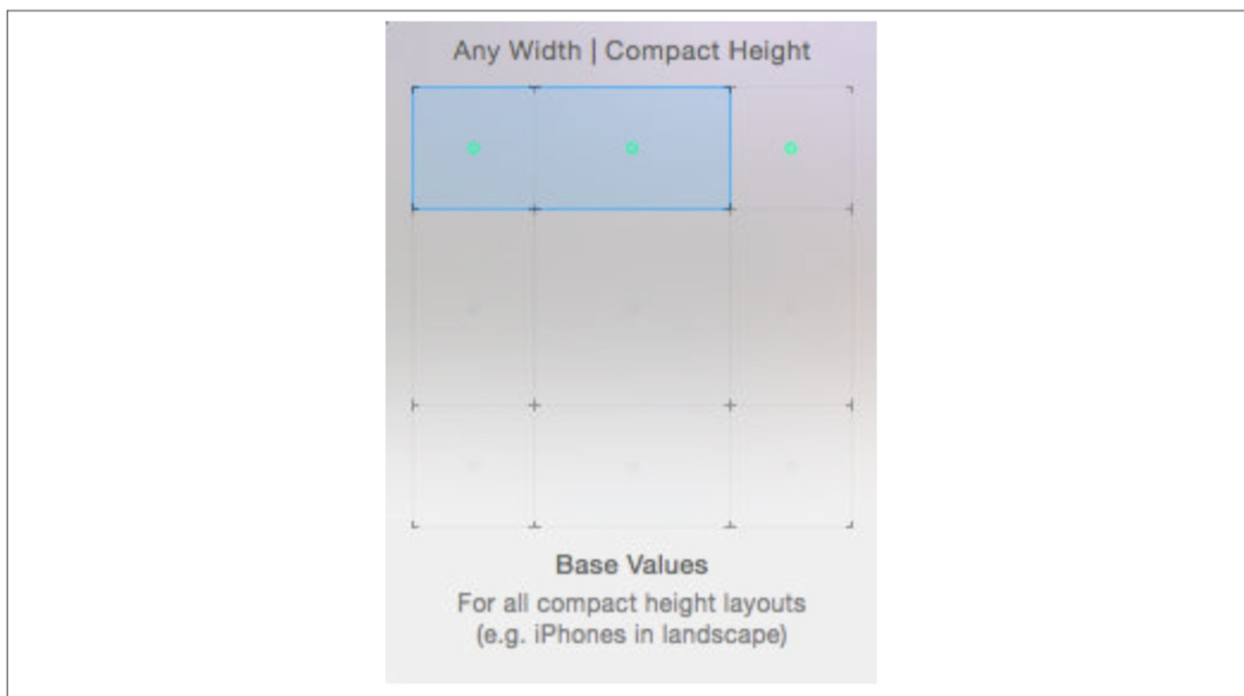
右侧出现了一个4英寸iPhone的预览图，点击左下角的加号，选择iPad，一个iPad的预览效果就出现在Previewer中了（见图7-31）。现在各个控件在iPad上的效果看起来好多了。点击iPad preview下方的弯曲箭头，就可以调整iPad的方向，虽然方向变化了，但是控件的位置还是放置正确，调整一下4英寸iPhone，方向变化后，界面看起来不太友好了，各个控件的位置似乎并不合适。



Exercise: Building More on the Passport App | Page 209

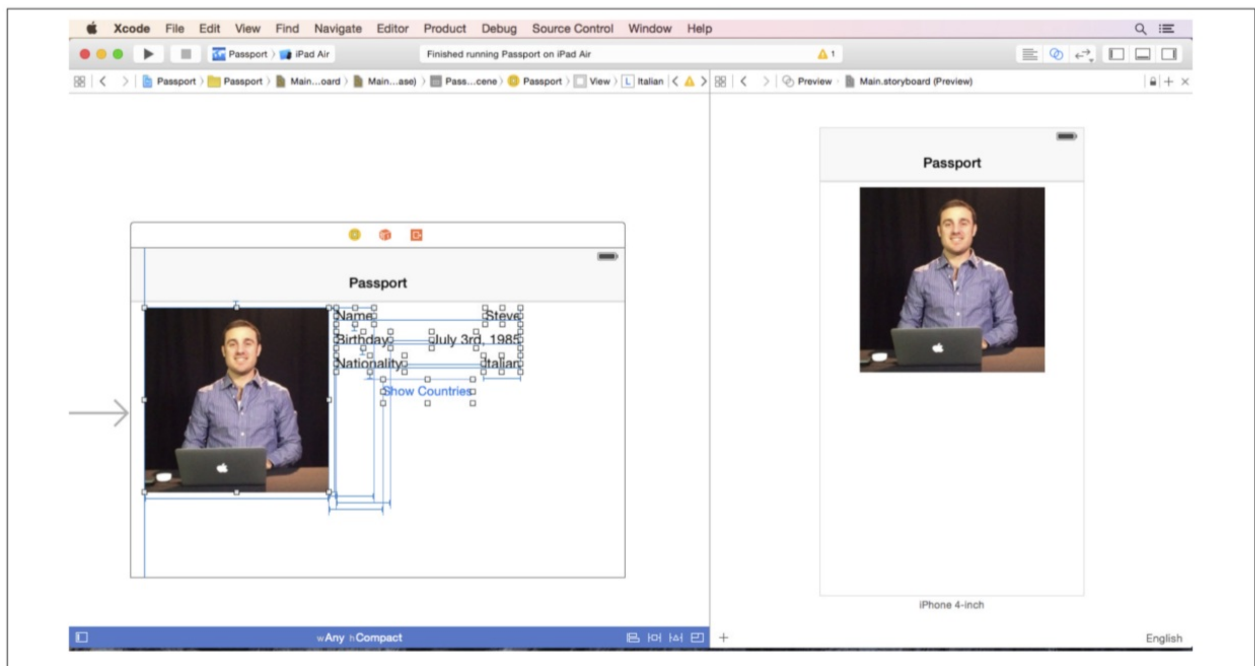
在最下方点击“Regular，Regular”，然后点击最上边一行格子的中间的那个格子，这时size class变成了“Any,Compact”（见图7-32）。这个size class是用来设计iPhone横屏的效果，这时界面会调整方向，有些控件会无法出现在界面中。为了解决这个问题，我们需要从顶部菜

单栏中点击Edit -> Select All，这样所有的控件都选中了，把Image View拖到界面的最顶部，所有的控件都显示出来了。

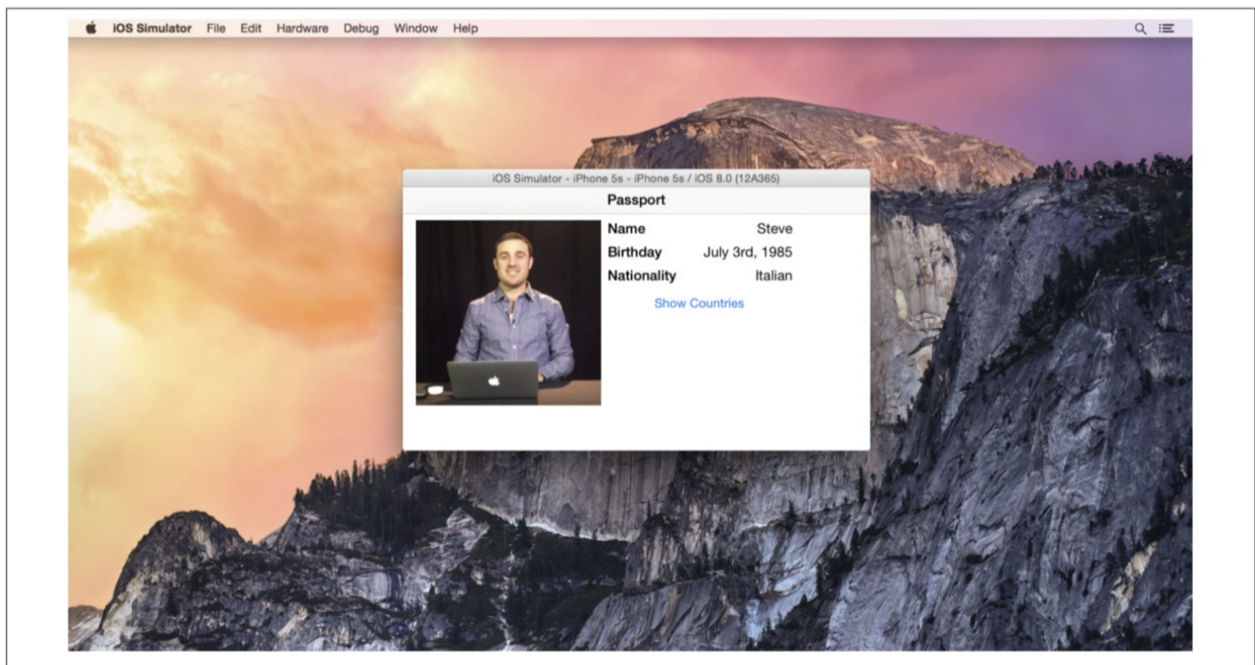


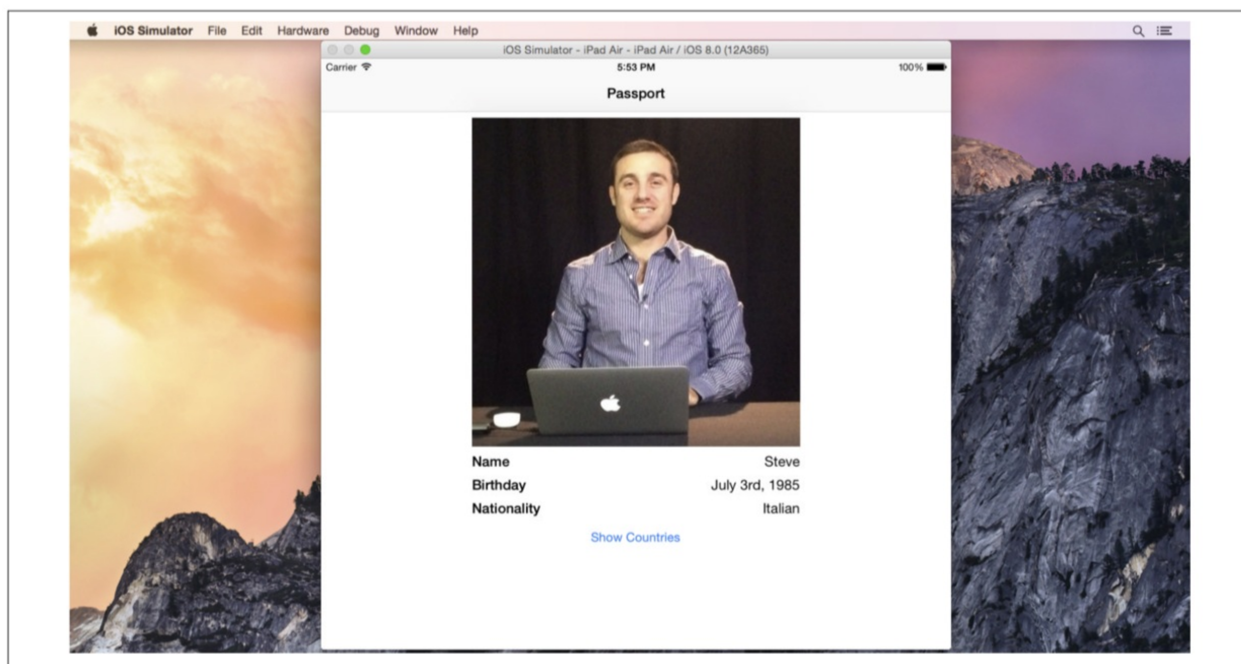
Page 210 | Chapter 7: Devices and Auto Layout

这个size class需要全新的Layout（见图7-33）。点击Resolve Auto Layout Issues按钮，选择最下边的Clear Constraints，这样就会清除这个size class下的所有约束条件。接着把Image View放到屏幕的左上方，直到左边和上方都出现了辅助线，选中Image View，点击Pin按钮，勾选上方方向竖线，选择Use Standard Value，左侧方向竖线，选择Use Current Canvas Value。勾选Width和Height选项，点击Add Constraints，把Label和Button放到图片右侧上方，直到上方出现了辅助线，以及Image View的右侧也出现了辅助线，点击Pin按钮，上方方向竖线选择Use Standard Value，左侧方向竖线，选择Use Current Canvas Value，点击Add Constraints。



所有的设置都已经完成了，可以对App进行测试了，模拟器选择iPhone 5s，点击Play按钮（Command+R），选择Hardware -> Rotate Right（Command+方向键）（见图7-34），在iPad Air上运行也没有问题（见图7-35）





> Exercise: Building More on the Passport App | Page 211

如果App没有按照你想要的结果运行，或者程序有了错误或警告，不要太担心，学习的最佳方式就是试错，熟能生巧，到我们的网站上下载示例代码，对比一下代码，多试几次，直到搞定这个程序为止。

> Page 212 | Chapter 7: Devices and Auto Layout

第八章：地图和位置

在这一章节中，你将会学到在你的app中如何使用iPhone中的GPS。你将会学会如何获得用户的位置，同时在地图上标注出位置。本章内容涵盖了 地图套件（Map Kit）和苹果公司提供的地图、方向框架。通过学习本章节的内容，你马上就可以搞定基于位置定位的应用。

提供用户的位置信息是开发iOS应用最激动人心的特性，在地图上显示用户的位需要两个步骤。

第一步用Core Location来收集用户的位置。Core Location是一系列类的集合，通过设备的GPS和蜂窝获取位置信息，还能借助WIFI获取用户信息。Core Location 是由苹果公司提供的众多frameworks中的一个。Frameworks是一组类的集合，为具体的某个任务而设计的一套工具。我们现在用Core Location举一个例子，Core Location是为了处理用户位置信息而设计的一个framework。Core Data是为了处理数据而设计的一个framework。这些framework是可选的，因此需要先把这些框架导入到工程当中后，才能使用这些框架。

第二步是在地图中标注出用户的位置。苹果公司提供了Map Kit框架，帮助我们绘制和管理地图。在Xcode 6中导入框架非常容易。首先Project Navigator中点击项目名称，点击名为Capabilities的tab选项按钮，向下滚动找到Maps。将地图的开关键处于On的状态，这时MapKit框架已经添加到工程中了。现在虽然能够在工程中找到MapKit，但是导入流程还没有完成。MapKit还需要导入controller file中。我们下列一行代码完成导入工作：

```
import MapKit
```

这样，MapKit类和协议就导入到controller文件中了。MapKit中的类和协议都是以MK开头的。

> Page 213

Core Location

正如之前所说的，Core Location是一系列查找用户位置的类的集合。Core Location中有三个检测用户位置的方法。第一个方法是Significant-Change Location。这个方法能够节约电池电量，它只在用户的位置明显改变时才会更新位置。第二个方法是Location Services，可以自主规定定位更新的规则。最后一种方法是Regional Monitoring方法，使用附近的地理区域边界或者Bluetooth beacons来定位。本书主要介绍第二种方法：Location Services，它是最常用到的方法。

更多信息请参考苹果公司的Location and Maps Programming Guide。

获取用户位置需要使用Core Location框架，当你把Maps capabilities开关切换成On（开）状态时，Xcode并没有自动导入Core Location框架，需要我们手动导入，请看以下四个步骤：

1. 点击Project Navigator上的蓝色工程图标；
2. Editor显示工程的详细信息，滑倒最下方；
3. 在Link Binary with Libraries下方点击Add；
4. 选择Core Location，然后点击Add。

Core Location框架就会添加到Project Navigator中，我们还需要在controller中写一行代码才能获取此框架：

```
import CoreLocation
```

> 
> 手机用户的位置非常耗费电量，它比其他的任务需要更多的电池电量和天线频率，所以确保你的App只有在需要位置时才获取位置，一旦获取到位置，就把这个功能关掉，如果将来还需要地理位置，可以使用定期更新功能

在获取用户位置之前，很重要的一件事是先检查定位服务是否可用。定位服务无法使用可能是由于以下几种情况：

- 用户在设置中关闭了Location Services（定位服务）。
- 用户禁止你的App使用Location Services（定位服务）。
- 设备处于飞行模式或者连接不了网络。

> Page 214 | Chapter 8 : Maps and Location

Core Location提供了名为 `locationServicesEnabled` 的方法来检查设备的定位服务是否可用， `locationServicesEnabled` 方法通过布尔类型返回值来确定定位服务是否可用，true可用，false不可用。

Requesting User Location

通过CLLocationManager类来请求用户位置。首字母CL代表Core Location。The location manager用于收集参数和开启定位服务。创建CLLocationManager对象和创建其他的对象类似。举例说明：

```
var locationManager: CLLocationManager = CLLocationManager()
```

CLLocationManager有一些属性是必须要设置的。

desiredAccuracy属性是枚举类型，枚举，就是用一个关键词代表一个数字。枚举有点像是多选题，你必须从选项中挑选出一个值来。desiredAccuracy属性有下面一些值：

`kCLLocationAccuracyBest` 最精准的定位，也是最消耗电量的选项
`kCLLocationAccuracyNearestTenMeters` 精准度在十米范围内
`kCLLocationAccuracyHundredMeters` 准确度在一百米范围内 `kCLLocationAccuracyKilometer` 精确度在一千米范围内 `kCLLocationAccuracyThreeKilometers` 精确度在三千米范围内

精准度越高，电量消耗越大。我们要选择能够满足最低要求的精准度级别。如果是像Google地图之类的App来追踪用户的位置，那么`kCLLocationAccuracyNearestTenMeters`或者`kCLLocationAccuracyHundredMeters`就可以满足我们的需求。如果App只需提供用户所在城市，像是Twitter中的定位，`kCLLocationAccuracyKilometer`或者`kCLLocationAccuracyThreeKilometers`就可以满足我们的需求。大多数情况下，一般不需要`kCLLocationAccuracyBest`。

设置desiredAccuracy属性的方法和设置其他对象的属性一样：

```
locationManager.desiredAccuracy = kCLLocationAccuracyHundredMeters
```

> Core Location | Page 215

CLLocationManager也需要delegate属性。CLLocationManage delegate遵循CLLocationManagerDelegate协议。无论何时出现了位置更新或者出现错误，delegate都会接收到警告。为了接收这些警告，delegate必须通过CLLocationManagerDelegate协议与警报保持沟通。controller必须声明遵循协议，将CLLocationManagerDelegate添加到类的顶部：

```
class ViewController: UIViewController, CLLocationManagerDelegate
```

想要接收定位警高，要使用`locationManager(_:,didUpdateLocations:)`方法。每当定位信息改变的时，这个方法就会被调用：

```
func locationManger(manager: CLLocationManager!,didUpdateLocations locations: [AnyObject]
    println("Location found")
}
```

想要在任何时候都能收到Core Location的错误警告，需要使用`locationManager(_: didFailWithError:)`方法：

```
func locationManager(manager: CLLocationManager!, didFailWithError error:NSError!) {
    println("Error!")
}
```

一旦遵循协议并使用其中的方法后，就必须要设置delegate的属性：

```
locationManager.delegate = self
```

在激活定位服务之前， 用户必须同意app使用用户的位置信息。定位服务有两种批准类型。第一种是requestWhenInUseAuthorization;授权App仅限前台运行的时使用位置信息。 第二种是requestAlwaysAuthorization. 授权App在前台还是后台运行都可以获取用户的位置信息，第二个授权都会给app提供追踪用户位置的能力。调用授权的方法是：

```
locationManager.requestWhenInUseAuthorization()
```

或

```
locationManager.requestAlwaysAuthorization()
```

获得授权后，调用 `startUpdatingLocation()` 方法开启定位服务：

```
locationManager.startUpdatingLocation()
```

这样，locationManager会根据相关要求开始追踪并返回用户位置信息。

> Page 216 | Chapter 8 : Maps and Location

`locationManager (_:, didUpdateLocations:)` 方法会提供CLLocation数组，按照出现前后顺序排列。数组中至少会有一个对象。数组中的每一个对象都是是一个CLLocation。CLLocation这个类为具体的位置整理组织CLLocationManager的位置数据。CLLocation跟踪地理坐标，海拔，速度，方向，甚至包括定位准确度。CLLocation拥有许多有用的属性：

`coordinate` CLLocationCoordinate2D, 纬度坐标和经度坐标 `altitude` 海拔高度，单位：米 `timestamp` 获取到数据时的时间和日期 `description` 用字符串的格式返回CLLocation，可以用print()打印出来

请牢记，一旦你获得了你需要的信息，必须停止定位服务功能。为了停止这些服务，在CLLocationManager中调用 `stopUpdatingLocation()` ：

```
manager.stopUpdatingLocation()
```

举个例子，在 `locationManager( manager: ,didUpdateLocations:)` 方法中获取位置后，常常会停止定位服务。之前在CLLocationManager中创建的那些变量非常适合处理目前的这种情况。



明白了！ iOS 8模拟器在模拟Core Location时会出现一些前后不一致的行为。如果定位服务没有调用，在Info.plist文件中添加三个键：NSLocationWhenInUsageDescription NSLocationAlwaysUsageDescription NSLocationUsageDescription 每个键对应的值设置成Always或者When in Use 这三个键值会帮助开启定位服务

Core Location | Page 217

Map Kit

Map Kit框架提供地图和方向，地图可以展示到街道级别的信息，3D建筑，卫星图像，或者将两者组合起来。地图自动响应缩小、放大、平移、倾斜等手势动作，还能在地图上标注点同时加上注解。

MKMapView

Map Kit提供MKMapView视图类来展示地图，MKMapView可以展示地图，管理用户的输入信息，展示自定义注释。

MKMapView也有一个delegate属性。和CLLocationManager的delegate属性一样，MKMapView的delegate也能接收updates。MKMapView delegate需要遵循MKMapViewDelegate协议。设置delegate的方法是，从Storyboard的Editor中，将Map View用Control拖动法拖动到Document Outline中的View Controller文字上，然后弹出一个菜单，点击菜单中的delegate，这样就在相关界面上设置好了delegate。

MKMapView有很多方便的属性和方法。举了例子，MKMapView不用添加任何代码就可以在地图上展示用户地理位置。我们把属性设置 `showsUserLocation` 为 `true`，就可以在地图上显示用户信息了：

```
myMapView.showsUserLocation = true
```

用户的位置将会在地图上用一个蓝点标注出来。

一般我们把用户所在位置设置为地图的中心点。如果想移动重新设置地图中心点，需要设置 `centerCoordinate` 属性， `centerCoordinate` 属性需要 `CLLocationCoordinate2D` ， `CLLocationCoordinate2D` 是经度和纬度的坐标，被打包成一个单独的变量。通过 `CLLocationCoordinate2DMake` 方法创建 `CLLocationCoordinate2D` ：

```
var coordinates: CLLocationCoordinate2D = CLLocationCoordinate2DMake(100,100)
```

有时候我们会在地图上放大位置，当 `region` 属性设置好后，放大后图像会自动调整。 `region` 属性需要 `MKCoordinateRegion` 对象，然而，大部分情况下，比起创建新的对象，编辑当前的region对象会更简单一些：

```
var updatedRegion: MKCoordinateRegion = myMapView.region
updatedRegion.span.longitudeDelta = updatedRegion.span.longitudeDelta * 2.0
updatedRegion.span.latitudeDelta = updatedRegion.span.latitudeDelta * 2.0
myMapView.region = updatedRegion
```

> Page 218 | Chapter 8 : Maps and Location

`longitudeDelta` 和 `latitudeDelta` 都是span的一部分，span是面积有多大，以centerCoordinate为中心可展示的宽度和高度。

Directions （方向）

Map Kit还能够在App中提供建议规划路线导航功能。MKDirections API可以根据苹果服务器的计算提供线路方向。有步行线路规划，驾驶线路规划，花费的时间，和其他可选的路线。地图上的每个点用MKMapItem表示，MKMapItem包含了地图上有关地点的所有信息，这些信息包括地图位置，坐标值，地点名称等数据。MKMapItem还能传到地图应用上，使用地图应用上更多高级功能。

创建MKMapItem最简单的方法是使用 `mapItemForCurrentLocation` 方法，这个方法获取用户的位置然后根据位置创建MKMapItem：

```
var mapItem: MKMapItem = MKMapItem.mapItemForCurrentLocation()
```

MKMapItem类有一些便利的属性。name属性是一个字符串，提供地点的描述性名称。phoneNumber属性也是字符串，存储这个位置的电话号码。URL属性存储位置的网址。

MKMapItem创建后，就可以轻松的把位置传递到地图应用上，使用导航功能。 `openMapWithItems: launchOptions` 方法可接收一个数组，数组中包括包括一至多个的MKMapItem。通过 `launchOptions` ，这么些items就会被映射到地图应用上。 `MKLaunchOptionsDirectionsModeKey` 让地图应用基于两个点来提供规划路线。

Plotting Points （绘制点、标注点）

苹果公司提供了一个在地图上绘制点的方法，叫annotations(注解)。annotations是可以定义一个地方或者一个点。它常常用于突出感兴趣的地点，提供更多细节。annotations也拥有一个可选标注气泡（optional callout bubble）。气泡代表一些位置的名字和地址那样的信息。气泡也是可点击的，可以像button（按钮）那样接收用户的动作。

annotations由两部分组成，注解对象（annotation object）和注解视图（annotation view）。annotation object是一个轻量级对象，管理annotation中的数据。annotation object是从MKPointAnnotation类中创建的。annotation view是从MKPinAnnotationView类中创建的。annotation view用来在地图上标注pin（大头针）。

> Map Kit | Page 219

三个步将annotation添加到MKMapView中。第一步是为感兴趣的地点创建一个MKPointAnnotation：

```
var point = MKPointAnnotation()
point.coordinate = CLLocationCoordinate2DMake(37.7756, -122.4193)
point.title = "San Francisco"
```

接下来，遵循MKMapView协议，回应 `mapView(_: viewForAnnotation:)` 方法，此方法可以回收利用annotation view，就像是table view中也有方法可以重复利用cell：

```
func mapView(mapView: MKMapView!, viewForAnnotation annotation: MKAnnotation!)-> MKAnnota
    var pin = MKPinAnnotationView(annotation: annotation, reuseIdentifier:"pinIdentifier"
    return pin
}
```

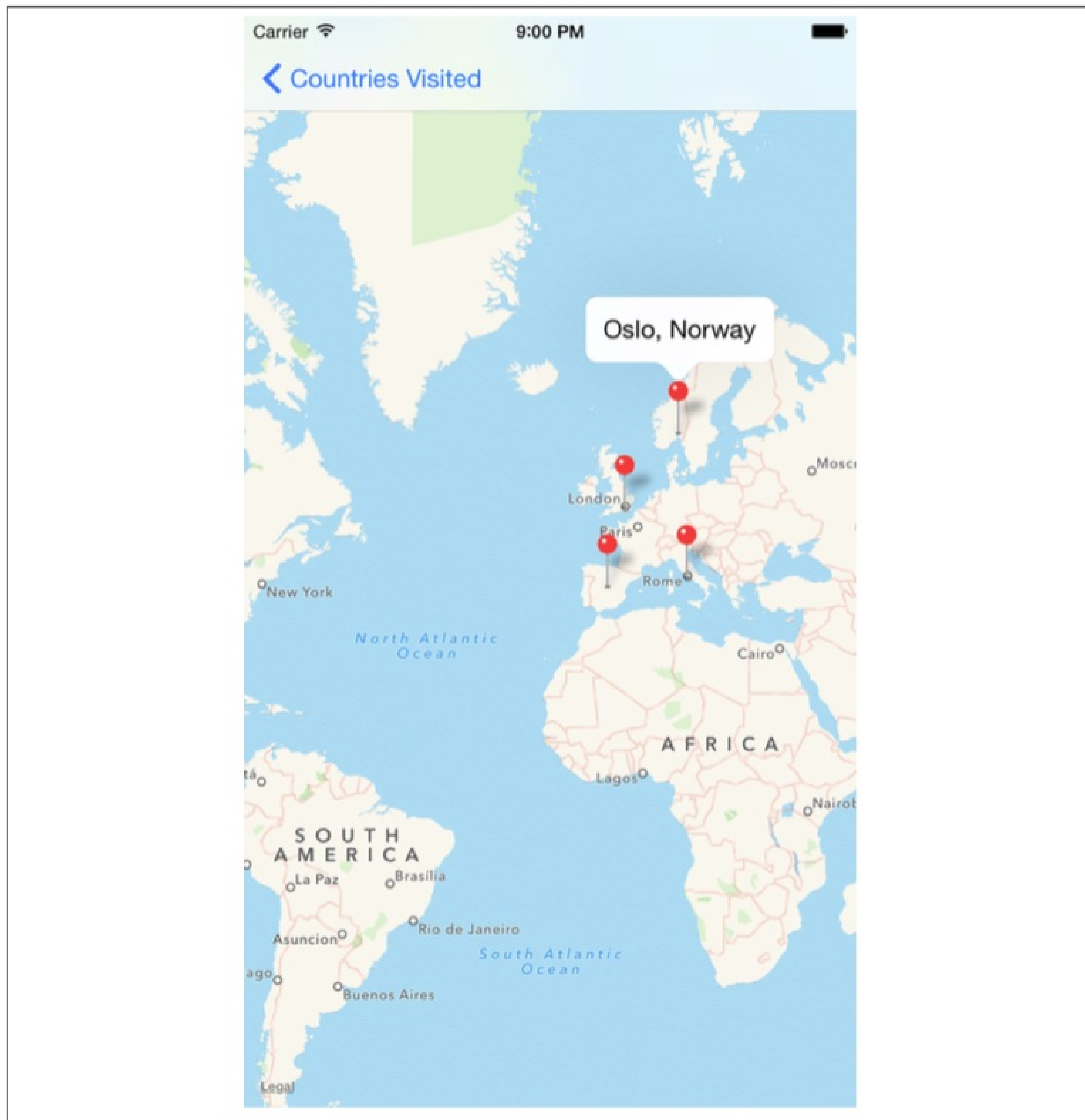
最后，调用 `addAnnotation` 方法，这样，就把annotation添加到地图中了：

```
mapView.addAnnotation(point)
```

现在，我们来搞定你第一个使用了地图App吧。

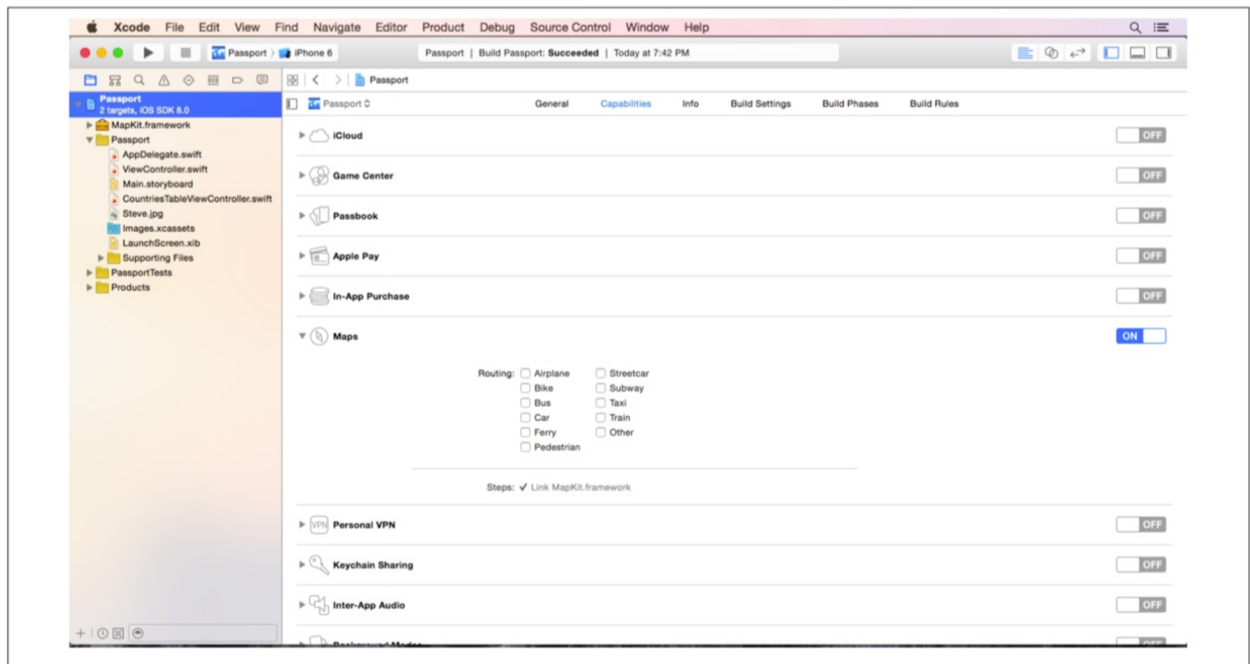
第八章练习 Adding Maps to the Passport App - 给护照App增加地图功能

在本章的练习中，我们要给前几章中的Passport App增加地图功能，添加一个Map View，在每个去过的国家上方都出现一个大头针（见图8-1）。当用户点击Countries Visited界面右上角的Map按钮时，出现地图界面。



Page 220 | Chapter 8: Maps and Location

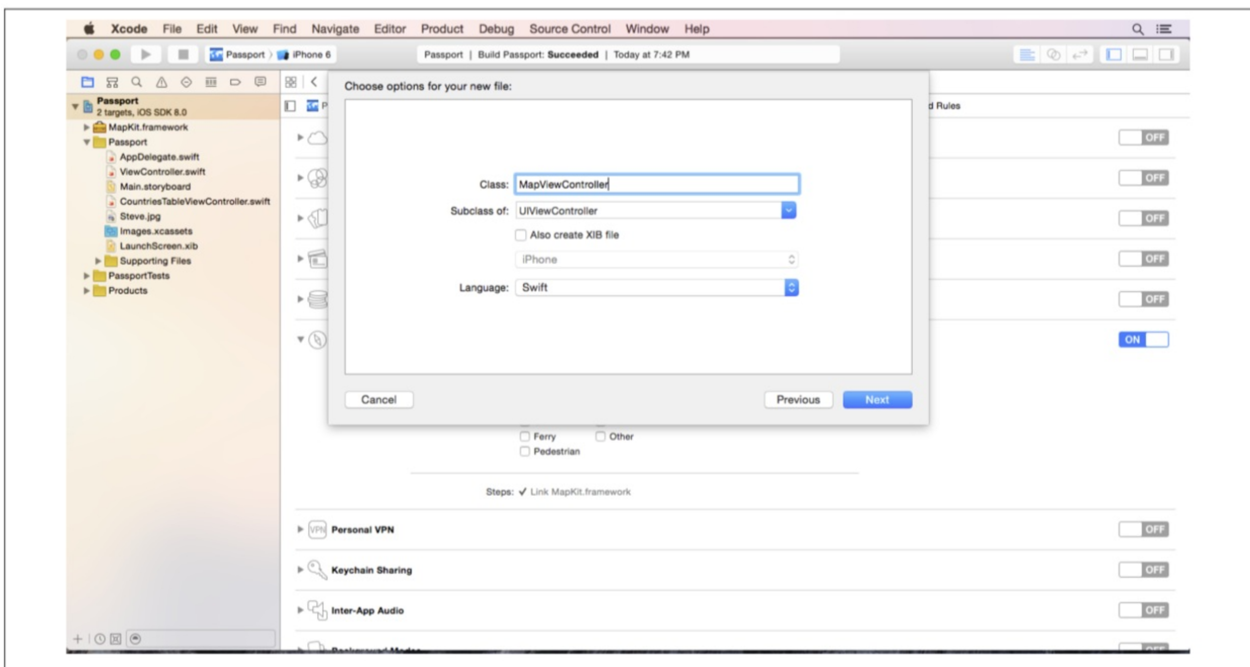
首先我们打开Passport.xcodeproj文件，点击Project Navigator中的工程名字Passport，点击Capabilities，往下滑，找到Maps，打开开关（见图8-2）。



Exercise: Adding Maps to the Passport App | Page 221

在Project Navigation中会出现MapKit.framework，接着选择顶部菜单栏File -> New -> File，选择iOS下的Cocoa Touch Class，点击Next。

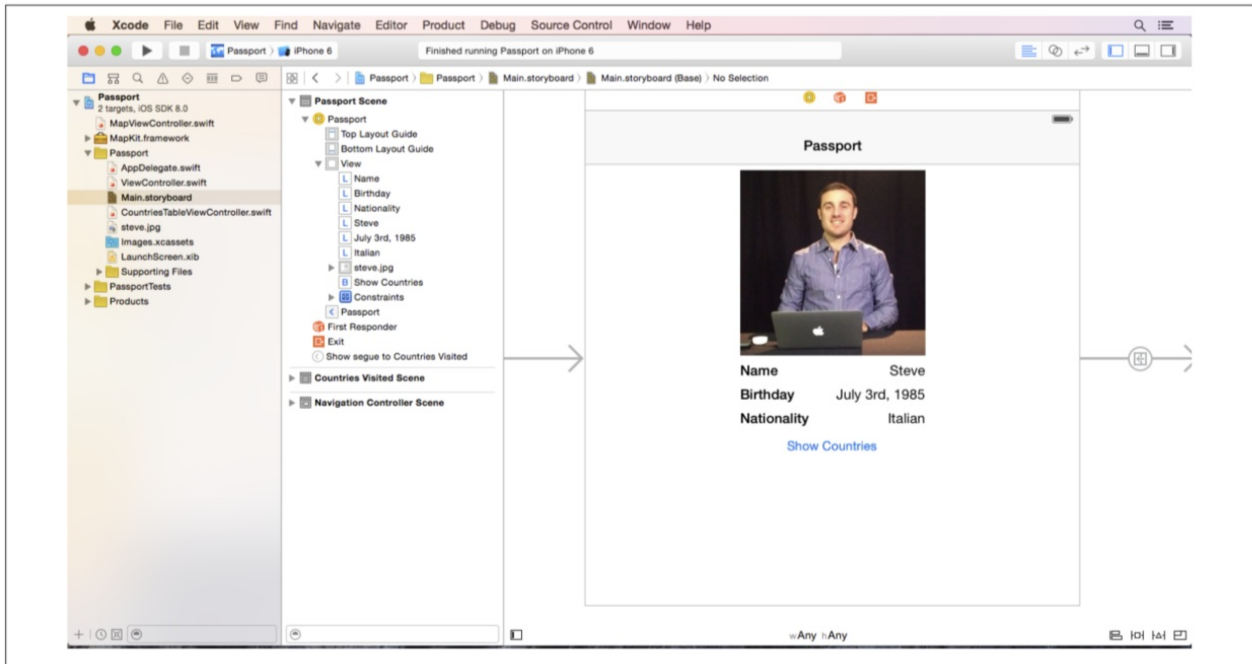
Subclass of选择UIViewController，Class名字为MapViewController（见图8-3），语言Swift，点击Next，保存在Passport文件夹中，点击Create。新文件MapViewController.swift出现在屏幕上。



Page 222 | Chapter 8: Maps and Location

选中文件中的绿色注释代码，删掉。注意不要误删最后的右大括号，删掉didReceiveMemoryWarning方法。

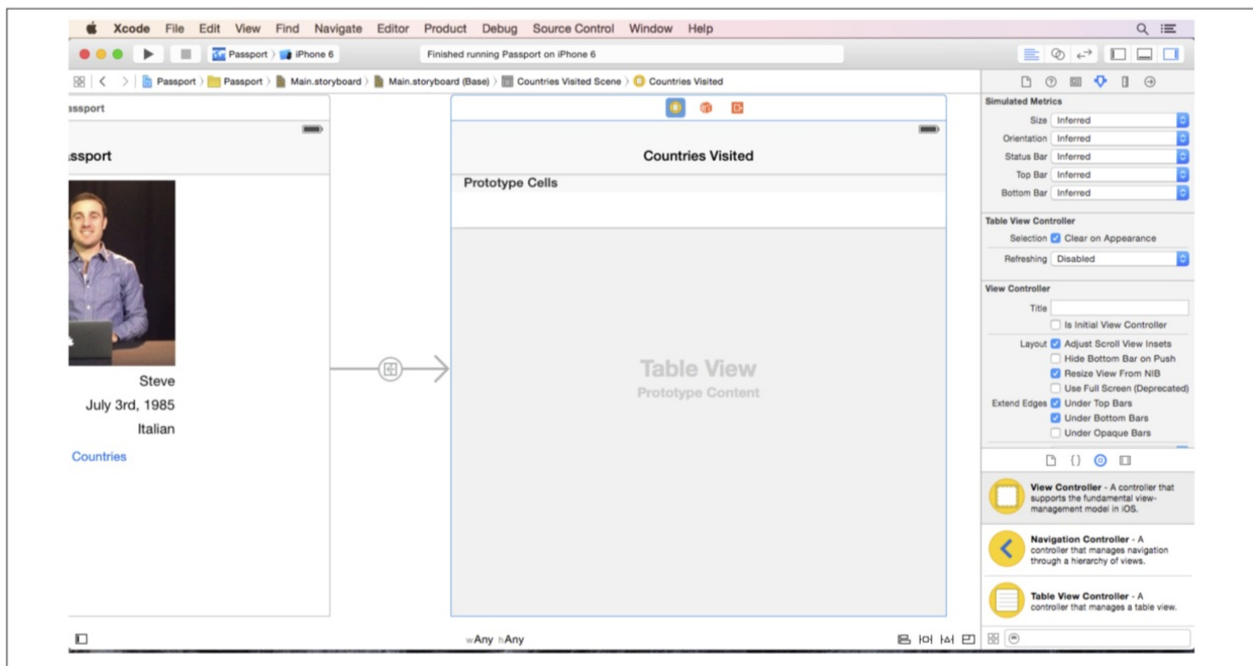
打开Main.storyboard（见图8-4），首先确认当前处于“Any,Any”这个size class下，如果不是，点击Size class，选择中间的那个格子（见图8-5）。





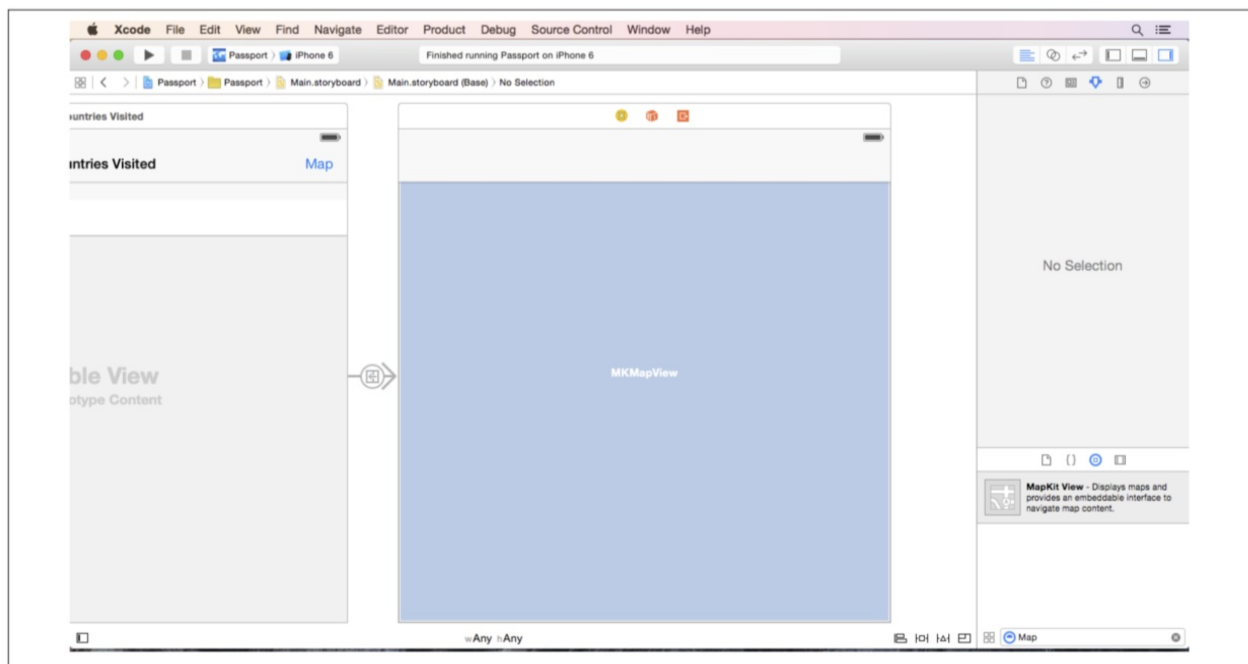
> Exercise: Adding Maps to the Passport App | Page 223

隐藏Project Navigator和Document Outline，打开Inspector，界面调整到Table View Controller Scene右侧（见图8-6）。



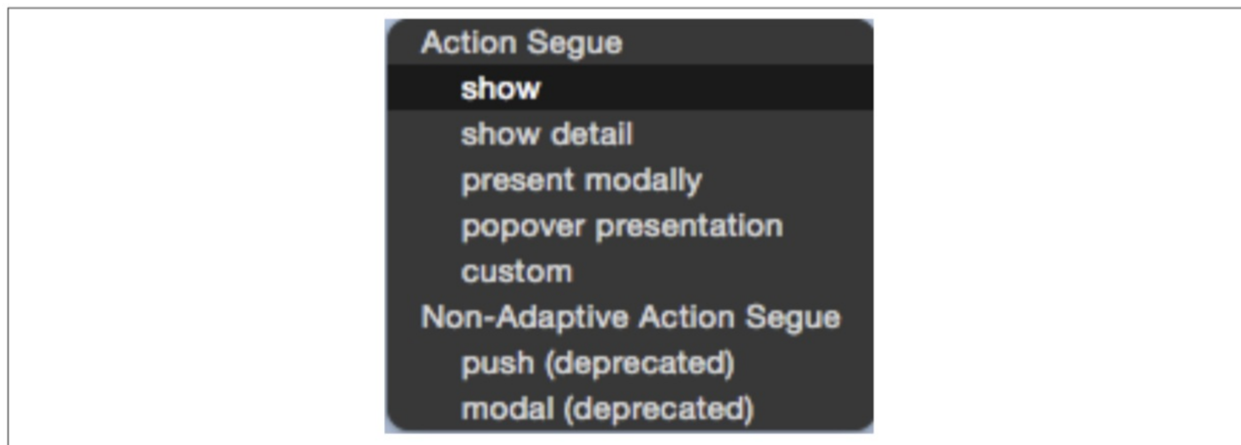
Page 224 | Chapter 8: Maps and Location

在Object Library中拖拽一个View Controller，放到Table View Controller Scene右侧，然后在Object Library搜索框中输入map，拖拽一个Map View到View Controller中（见图8-7）。把Map View四周拖拽变大直到覆盖整个View。



选中Map View，从顶部菜单栏选择Editor -> Pin -> Leading Space to Superview；再次选中Map View，从顶部菜单栏选择Editor -> Pin -> Trailing Space to Superview；再次选中Map View，从顶部菜单栏选择Editor -> Pin -> Top Space to Superview；再次选中Map View，从顶部菜单栏选择Editor -> Pin -> Bottom Space to Superview。这样Map View就能够适配任何尺寸的设备了。

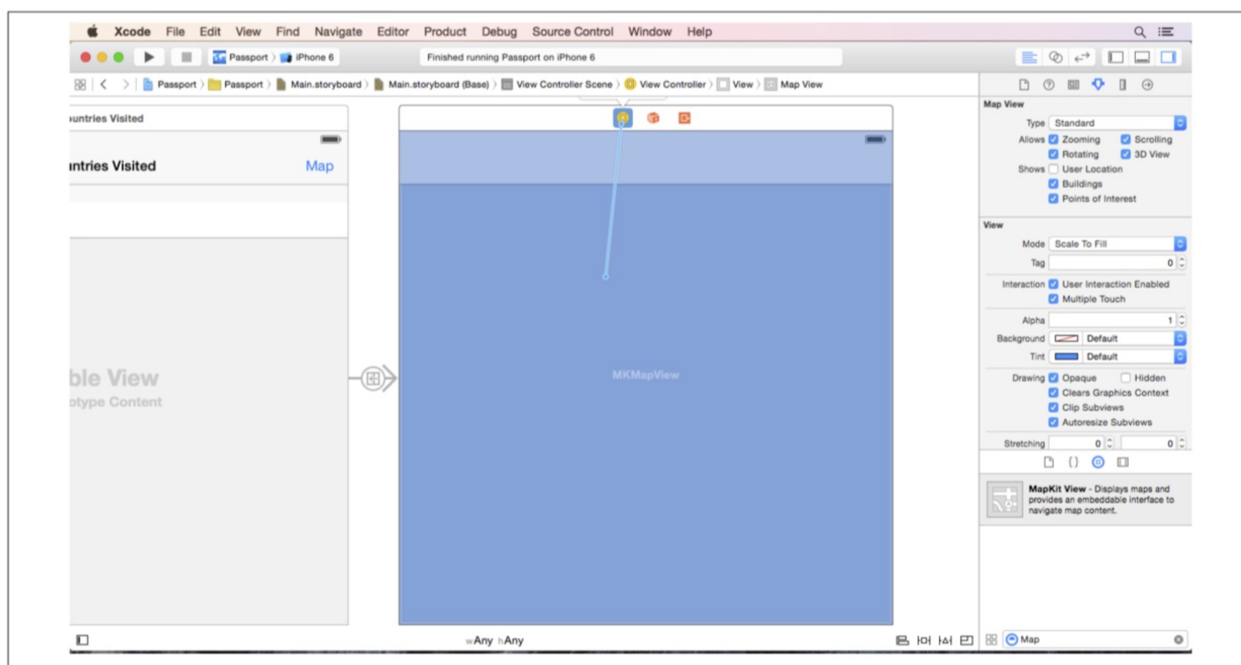
清空Object Library的搜索框，然后输入BarButton，选择Bar Button Item拖拽到Table View Controller Scene中，双击Bar Button Item，输入Map。按住Control键，点击Map按钮，拖拽到刚刚设置的有Map的View Controller上，弹出一个选项框，选择show选项（见图8-8）。



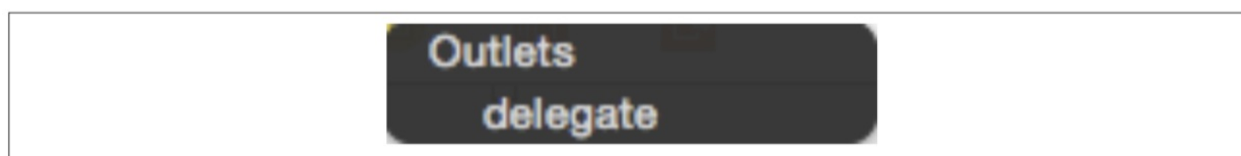
在Map scene中，出现了一个navigation bar。

> Exercise: Adding Maps to the Passport App | Page 225

点击其中的MKMapView，Control拖拽法拖到上方的黄色圆圈（见图8-9）。

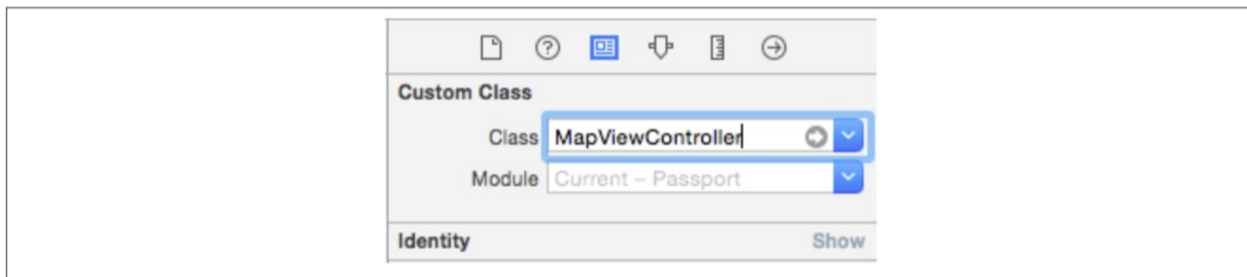


弹出一个小的菜单，选择delegate选项（见图8-10）。这样，MapViewController.swift文件能够收到MKMapView的委托通知（delegate update）了。

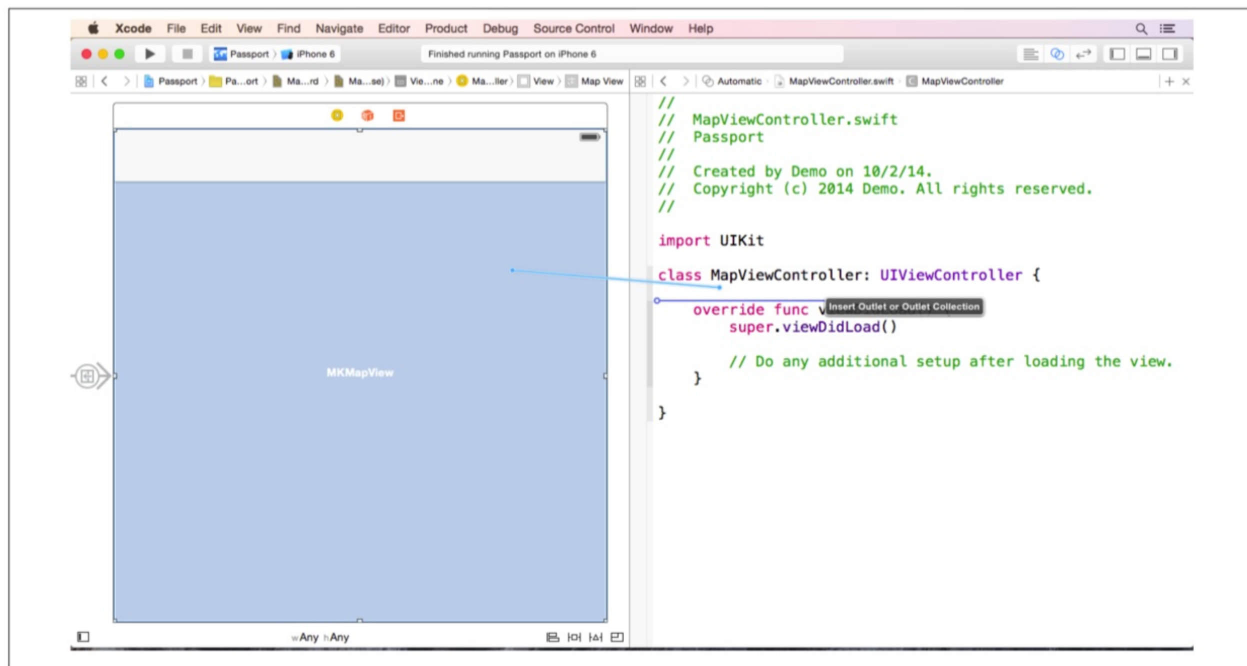


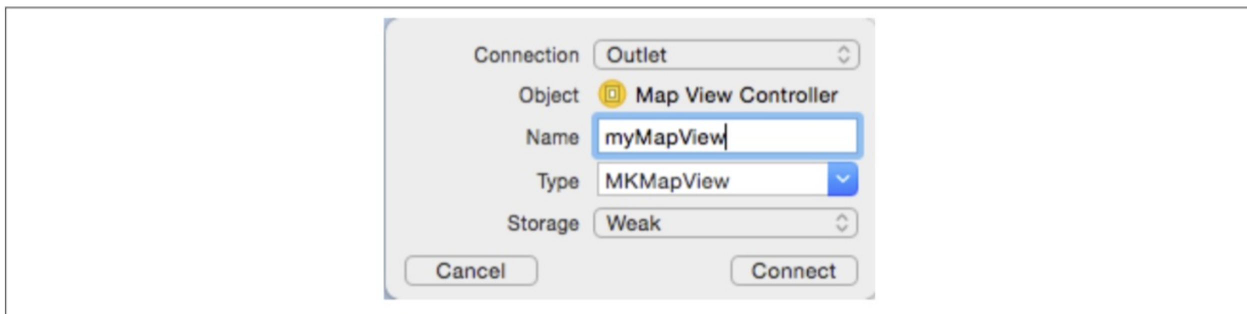
> Page 226 | Chapter 8: Maps and Location

storyboard的设置几乎快要完成了。必须把Map scene和MapViewController.swift文件关联起来。点击Map scene上的黄色圆圈，接着打开Identity Inspector，工具栏中点击第三个图标，在Class这输入MapViewController（见图8-11）。这样，Map scene和MapViewController.swift文件就关联起来了。



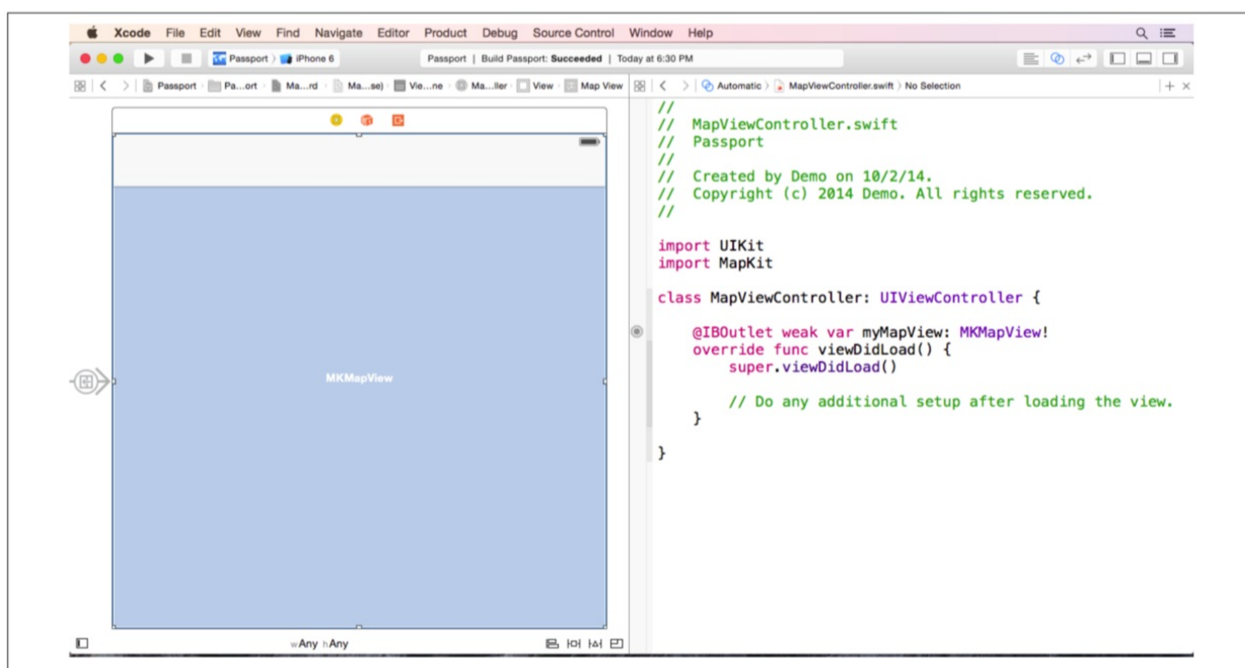
隐藏Inspector打开Assistant Editor，点击Assistant Editor上方的Preview，选择Automatic，Control拖拽法把MKMapView拖到MapViewController.swift文件中（见图8-12），弹出菜单（见图8-13）。





Exercise: Adding Maps to the Passport App | Page 227

Name一栏输入myMapView，点击Connect，创建了一个IBOutlet连接。不过这里出现了红色报错，出现错误是因为Map Kit framework还没有引入到MapViewController类中，输入import MapKit即可（见图8-14）。



> Page 228 | Chapter 8: Maps and Location

这时报错提示消失了。不管什么时候map出发事件，MapViewController都会收到delegate update。当然我们还需要让MapViewController使用MKMapView Delegate协议，修改增加下列代码：

```
class MapViewController: UIViewController, MKMapViewDelegate {
```

这样声明MapViewController会遵守MKMapView Delegate协议，把鼠标放到viewDidLoad方法中的super.viewDidLoad()下方。

在viewDidLoad方法中，你需要创建四个MKPointAnnotation对象，这四个对象能存储大头针需要出现的位置和显示文案。在viewDidLoad方法中添加下列代码：

```
let italy = MKPointAnnotation()
italy.coordinate = CLLocationCoordinate2DMake(41.8947400,12.4839000)
italy.title = "Rome, Italy"

let england = MKPointAnnotation()
england.coordinate = CLLocationCoordinate2DMake(51.5085300, -0.1257400)
england.title = "London, England"

let norway = MKPointAnnotation()
norway.coordinate = CLLocationCoordinate2DMake(59.914225, 10.75256)
norway.title = "Oslo, Norway"

let spain = MKPointAnnotation()
spain.coordinate = CLLocationCoordinate2DMake(40.41694, -3.70081)
spain.title = "Madrid, Spain"
```

> Exercise: Adding Maps to the Passport App | Page 229

这样就创建了MKPointAnnotation对象。第一个行代码把常量italy设置为一个MKPointAnnotation对象，coordinate属性可以设置具体的位置，CLLocationCoordinate2DMake方法获取位置的经度和纬度，然后转换成一组coordinate。最后title属性是郭靖和城市的名字，当用户点击地图上的大头针时，出现title。

四个MKPointAnnotation对象要添加到Map中，不过首先需要添加到数组中，创建一个数组，存储这四个位置，把下列代码添加到span MKPointAnnotation后面：

```
let locations = [italy, england, norway, spain]
```

这样创建了名为locations的数组，数组中有四个MKPointAnnotation对象。 添加到map中方法使用addAnnotations()方法，在之前的代码下方添加下列代码：

```
myMapView.addAnnotations(locations)
```

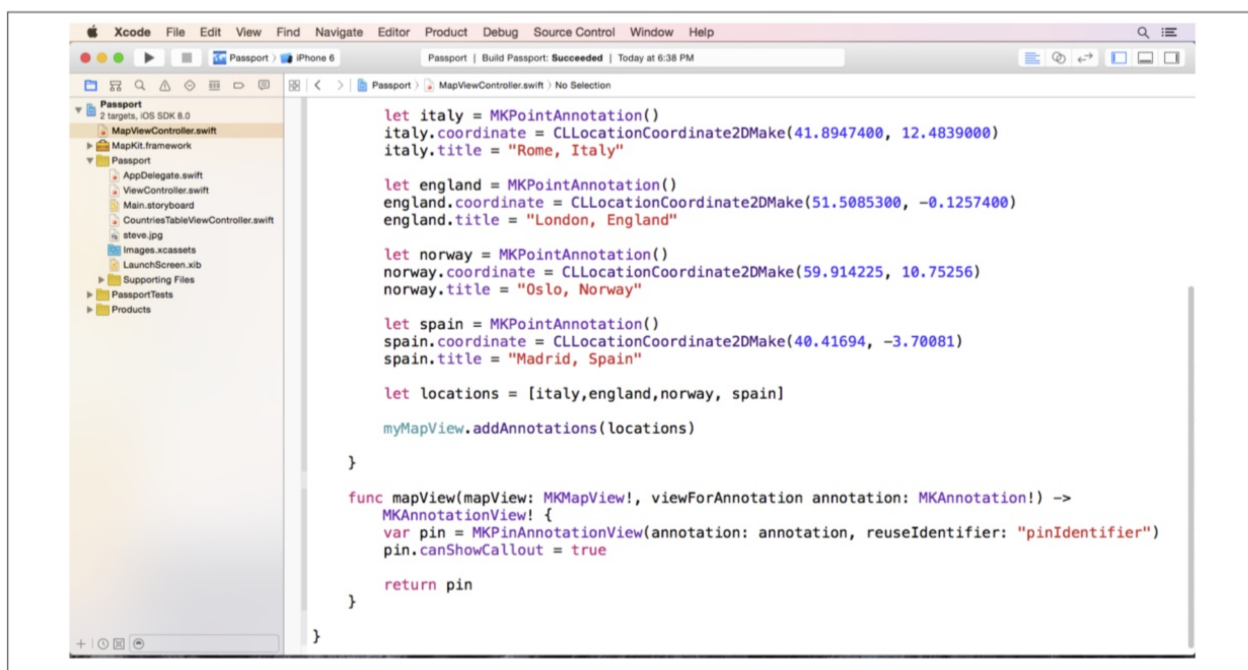
这行代码能够发送消息给myMapView，让其把italy,england,span和norway的地点添加到地图中。然而，我们还需要做一件事，所有的MKPointAnnotation都必须与MKPinAnnotationView配对，我们用mapView(_: viewForAnnotation:)方法来配对。把鼠标光标放到viewDidLoad方法外面的下方，输入下列代码：

```
func mapView(mapView: MKMapView!, viewForAnnotation annotation: MKAnnotation!)-> MKAnnota

}
```

当一个MKPointAnnotation添加到地图中后，这个方法回应MKMapView Delegate触发事件。这样MKPointAnnotations和MKPinAnnotationViews之间完成匹配工作。在mapView(_: viewForAnnotation:)方法里面，添加下列代码（见图8-15）：

```
func mapView(mapView: MKMapView!, viewForAnnotation annotation: MKAnnotation!)-> MKAnnota
    var pin = MKPinAnnotationView(annotation: annotation,reuseIdentifier: "pinIdentifier"
    pin.canShowCallout = true
    return pin
}
```



Page 230 | Chapter 8: Maps and Location

每次annotation添加到地图中时，这个方法都会被调用。第一行创建变量pin，赋值MKPinAnnotationView。 MKPinAnnotationView创建成功，annotation属性设置为annotation参数。

当前的MKPinAnnotation有参数值annotation，这样可能会造成困惑，然而，我们经常见过统一的名字使用两次代表不同的变量。关键是认识到annotation属性和annotation参数值的去边，知道两者的区别才是关键。annotation属性，在MKPinAnnotationView中可用，用来定位地图大头针的位置。而annotation参数值，是MKPinAnnotation匹配MKPinAnnotationView。

MKPinAnnotationView中的reuseIdentifier设置为pinIdentifier，这样可以回收利用MKPinAnnotationView，尤其是地图中的大头针特别多时，回收利用会有明显的优势。最后，MKPinAnnotationView的canShowCallout属性值设置为true。这样当大头针被用户点击时，能够出现泡泡效果。

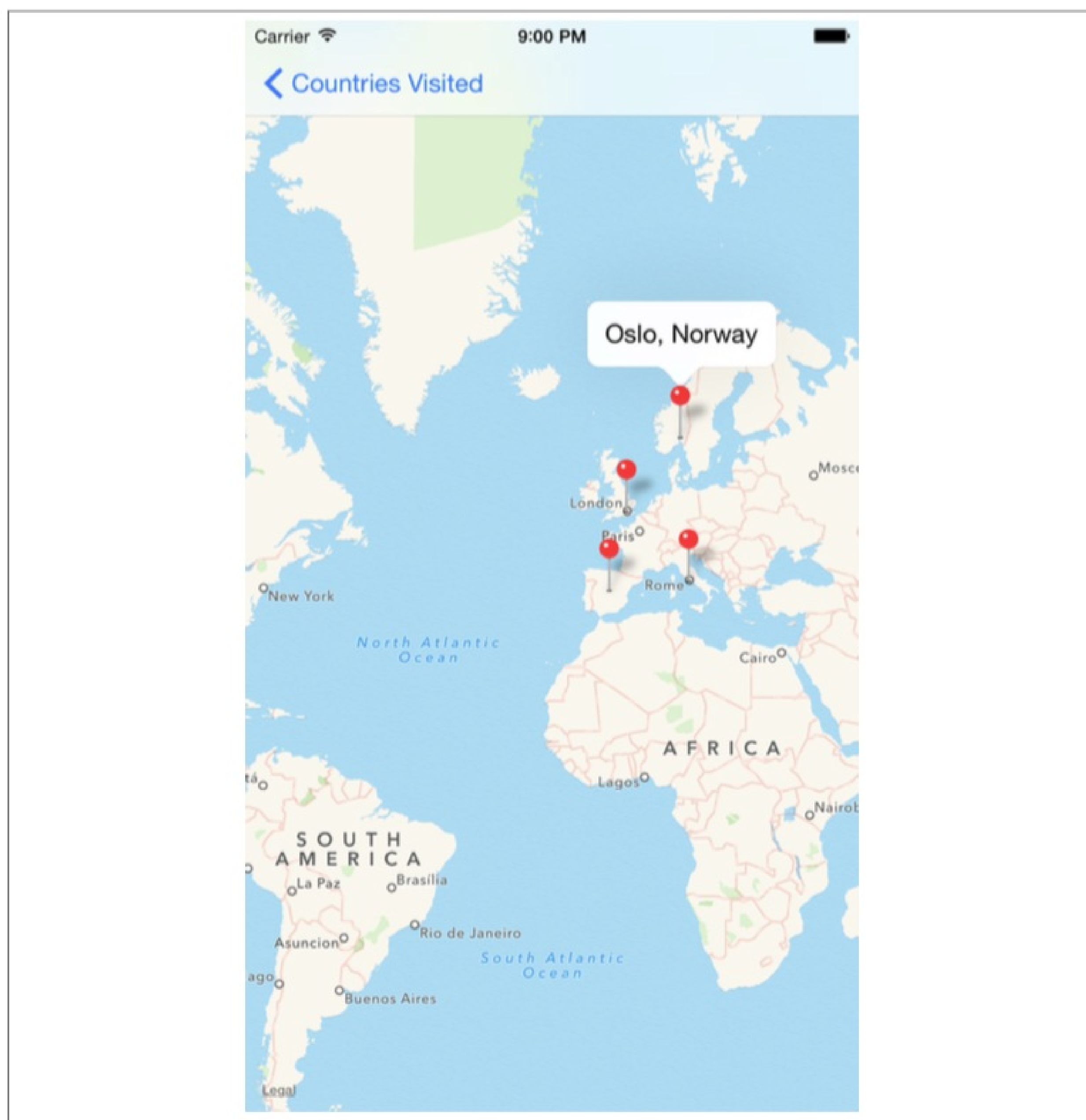
把鼠标光标放到myMapView.addAnnotations(locations)下方，添加下列代码：

```
var myRegion = MKCoordinateRegionMakeWithDistance(italy.coordinate,5500000,5500000)
myMapView.setRegion(myRegion, animated: true)
```

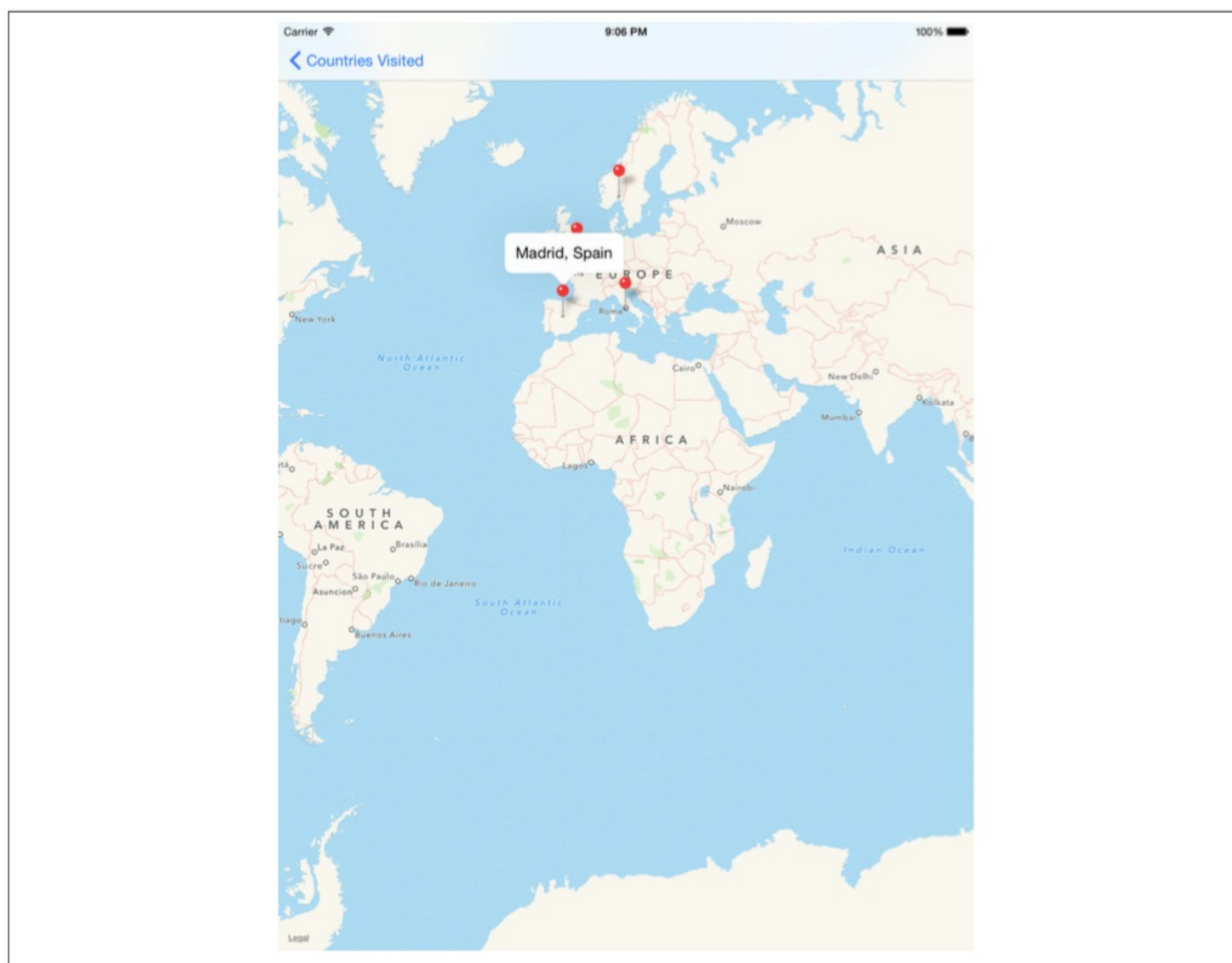
Exercise: Adding Maps to the Passport App | Page 231

第一行是创建名为myRegion的变量，设置只为MKCoordinateRegion。可以用MKCoordinateRegionMakeWithDistance()方法创建MKCoordinateRegion。这个方法首先找到一个点，然后围绕这个点画出一块区域，这个区域就是地图呈现的区域。

这第二行代码是给map发送消息，告诉Map把区域设置成新创建的myRegion，true表示可以放大缩小，运行程序，这次模拟器选择iPhone 6 Plus（见图8-16）。



现在当App启动后，大头针就会自动出现在对应的国家上了。结束运行，选择iPad Air模拟器，运行看一下效果（见图8-17）。



> Page 232 | Chapter 8: Maps and Location

如果App没有按照你想要的结果运行，或者程序有了错误或警告，不要太担心，学习的最佳方式就是试错，熟能生巧，到我们的网站上下载示例代码，对比一下代码，多试几次，直到搞定这个程序为止。

> Exercise: Adding Maps to the Passport App | Page 233

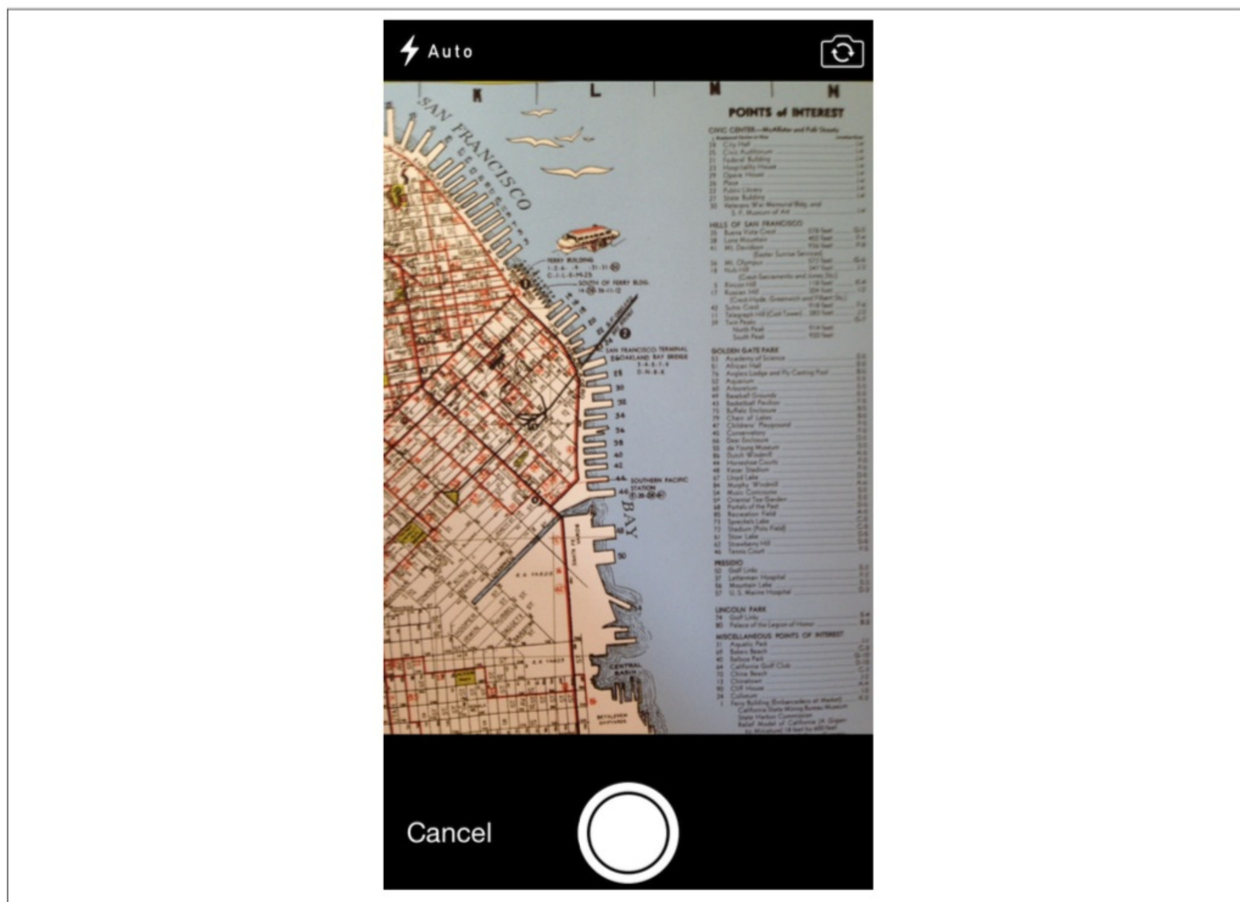
第九章：相机、相册和社交网络

在这一章节中，你将学会如何在应用中添加相机功能，还会学会如何从相册中获取图片和视频，最后，将学会如何给应用增加分享到Facebook和Twitter的功能。这一章将教会你Swift语言中最重要一部分，这样你能够更快的开发你的App。苹果公司提供了简单快速的方法来把相机图片和分享设计网络集成到一起。

UIImagePickerController

苹果提供了一个简单的类来处理图片和视频。图9-1中的*UiImagePickerController*看起比较眼熟，被广泛的使用在很多iOS App中。*UiImagePickerController*能够让相机中的景象直接显示在App中，*UiImagePickerController*还能够展示相册中的图片，让用户选择图片或者视频。

*UiImagePickerController*是一个简单的界面，直接在另外的view controller（视图控制器）中模式展示（presented modally）（presented modally是一个Segue类型）。一个modal view controller直接在当前的view controller中显示，有点像是弹出框。modal view controller会从下方滑出，而且只能由用户关闭才能返回之前的界面。modal view controller只能用在临时的交互或者短期的交互中。



> Page 235

创建*UIImagePickerController*对象和创建其他对象一样，首先声明一个变量，然后调用初始化：

```
var imagePicker = UIImagePickerController()
```

Cameras（相机）

在使用*UIImagePickerController*之前需要先检查一下当前设备的相机是否可用，在某些情况下，设备虽然有相机但是是不能用的。检查设备相机是否可用的代码：

```
if UIImagePickerController.isSourceTypeAvailable(.Camera) {
    //Camera is Available
} else {
    //Camera not Available
}
```

如何相机可用，则可以设置*UIImagePickerController*来调用相机获取拍摄图片或者视频。把*UIImagePickerController*设置成相机模式，需要把 `sourceType` 属性设置为 `.Camera` ：

```
imagePicker.sourceType =  .Camera
```

> Page 236 | Chapter 9 : Camera, Photos, and Social Networks

很多iOS设备有前后两个摄像头。检查前置摄像后可用还是后置摄像头可用，我们使用 `isCameraDeviceAvailable` 这个方法。把摄像头的位置传给 `isCameraDeviceAvailable` ，然后这个方法就会返回true或者false：

```
if UIImagePickerController.isCameraDeviceAvailable(.Front) {
    //Front Camera Available
} else {
    //Front Camera Not Available
}

if UIImagePickerController.isCameraDeviceAvailable(.Rear) {
    //Rear Camera Available
} else {
    //Rear Camera Not Available
}
```

如何当前设备没有可用的摄像头，*UIImagePickerController*可以让用户从相册中选择图片或者视频。把 `sourceType` 属性设置成 `.PhotoLibrary` 就可以展示选择图片界面了：

```
imagePicker.sourceType =  .PhotoLibrary
```

在iOS模拟器中测试App有时候会有局限性，因为模拟器没有摄像头。永远不要假设设备有相机，总是检查是否有摄像头。为了测试摄像头功能，你必须在真机上运行应用，在真机上运行应用的知识我们将会在第十章进行详细的介绍。

Media Types （媒体类型）

媒体的类型可以设置为图片、视频、和图片+视频。默认设置是图片+视频。想要改变这个设置，需要引入Mobile Core Services framework（Mobile Core Services框架）。

首先点击Project Navigator中的工程名称，显示工程的详细信息，然后滑到底部，找到Linked Frameworks and Libraries这部分，点击左下角的加号，在搜索框中输入**MobileCoreServices**，选择*MobileCoreServices.framework*，点击Add按钮。

这样*MobileCoreServices.framework*文件就添加到你的Project Navigator中了，接着你需要在代码中添加这行代码：

```
import MobileCoreServices
```

有了这行代码，*MobileCoreServices.framework*类就会添加到你代码中，你可以使用这个框架了。下面是每个媒体类型的关键词：

> UIImagePickerController | Page 237

`kUTTypeImage` 相片和图片 `kUTTypeMovie` 电影和视频 用数组来设置 `mediaTypes` 属性，可以存数合适的数值。例如：

```
imagePicker.mediaTypes = [kUTTypeImage]
//只能拍摄或选择图片
imagePicker.mediaTypes = [kUTTypeMovie]
//只能拍摄或选择视频
```

Editing（编辑）

苹果甚至提供了缩放图标和修剪视频的功能，叫做*editing controls*，把 `allowsEditing` 属性设置为 `true` 就能开启这些功能了。例如：

```
imagePicker.allowsEditing = true
```

这样就能进入编辑界面，你还能获取到编辑后和编辑前的媒体。

Delegates （委托）

*UiImagePickerController*提供了一些用户基本交互操作的委托更新（delegate updates）（这里不确定delegate updates的翻译是否正确）。例如当用户存储一个新的媒体时，就触发了一个delegate updates。为了能够收到这些updates，我们需要把*UiImagePickerController*委托到当前的view controller，使用self关键词来表示当前的view controller。例如：

```
imagePicker.delegate = self
```

当然，我们还要保证当前的view controller已经遵从了*UiImagePickerController*协议，遵从协议的方法很简单，就在当前的view controller中添加这行代码：

```
class ViewController: UIViewController, UIImagePickerControllerDelegate {
```

因为*UiImagePickerController*继承自*UINavigationController*，所以还要遵从*UINavigationController*协议，这份协议提供*UINavigationController*如push和pop等事件的updates，你的view controller必须要遵从协议，但是协议中方法（methods）都是可选的。详见代码：

```
class ViewController: UIViewController, UIImagePickerControllerDelegate, UINavigationCont
```

Page 238 | Chapter 9 : Camera, Photos, and Social Networks

Working with Images

`UIImagePickerControllerDelegate` 有两个方法来处理媒体，第一个方法是 `imagePickerController(_: didFinishPickingImage)` ，当相机拍摄了一张图片或者从相册中选择了一张图片后，这个方法就会触发delegate。如果用户是创建或选择的视频，那么这个方法不会被调用。这个方法提供了UIImage参数，来包括新创建的图片。为了能够出发这个事件，我们需要添加下列代码：

```
func imagePickerController(picker: UIImagePickerController!,didFinishPickingImage image:

}
```

Working with Multiple Media Types

第二个方法更高级一些，是 `imagePickerController(_:, didFinishPickingMediaWithInfo _:)` ，当用户创建选择了图片视频，这个方法会触发委托。如果 `sourceType` 设置为 `.PhotoLibrary` ，那么这个方法会在用户选中图片或视频后被调用；如果 `sourceType` 设置

为 `.Camera` ，用户拍摄了图片或视频并且确认保存后，这个方法才会被调用。 `info` 参数值为新图片视频提供了额外的信息。 添加下列代码可激活这个事件：

```
func imagePickerController(picker:UIImagePickerController,didFinishPickingMediaWithInfo i
    //media selected
}
```

`info` 参数还包括 `mediaType` 变量，在获取图片视频之前，需要先检查 `mediaType` ， 获取 `info` 参数中的键（key） `UIImagePickerControllerMediaType` 就可以检查 `mediaType` 了：

```
var mediaType = info[UIImagePickerControllerMediaType]
```

`info` 字典类型参数里面包含了很多关于新拍摄或选择图片视频的信息，下面这些键（key）将会提供所选内容的具体值：

`UIImagePickerControllerMediaType` Media type媒体类型， 类如：kUTTypeImage 或者 kUTTypeMovie

`UIImagePickerControllerOriginalImage` 原始未裁剪的图片

`UIImagePickerControllerEditedImage` 编辑过的图片，只有 `allowsEditing` 设置为true

> UIImagePickerController | Page 239

`UIImagePickerControllerCropRect` 将原始图片矩形裁剪

`UIImagePickerControllerMediaURL` 视频在本地文件中的路径（仅限视频）

`UIImagePickerControllerReferenceURL` 用于高级视频框架的URL（URL used with advanced video framework）

`UIImagePickerControllerMediaMetadata` 仅限图片，字典中图片的元数据（Photos only, dictionary full of metadata for image）

所有的这些键都能获取到媒体的更多信息，最常用的键是 `UIImagePickerControllerMediaType` ， `UIImagePickerControllerOriginalImage` ，和 `UIImagePickerControllerMediaURL` 。

`mediaType` 覆盖后，运行if语句来检测是图片还是视频（这话原话是Once the mediaType has been recovered, run it through an if statement to detect if itis a video or a photo.我不知道recovered在这里应该如何翻译啊……）。例如：

```
var mediaType = info[UIImagePickerControllerMediaType! as NSString]
if mediaType == kUTTypeImage as NSString {
    //photo
} else if mediaType == kUTTypeMovie {
    //video
} else {
    //error/missing
}
```

Images with didFinishPickingMediaWithInfo

如果 `mediaType` 的类型是图片，那么图片的会直接传入到 `info` 参数中，新图片会直接从字典中提取出来放入到UIImage，UIImage这个类是用来存储图片的，UIImage的图片传给UIImageView，UIImageView就像是相片的相框，它能把图片装裱起来，图片也能随时更换UIImageView在用户界面上展示UIImage的内容。例如：

```
var myImage = info[UIImagePickerControllerOriginalImage] as UIImage
```

把图片放入UIImageView的方法：把UIImageView的 `image` 属性设置为UIImagePickerControllerOriginalImage的键：

```
imageView.image = myImage
```

> Page 240 | Chapter 9 : Camera, Photos, and Social Networks

Video in didFinishPickingMediaWithInfo

如果 `mediaType` 的类型是视频，视频不会直接存储到字典中，而是存储视频文件的路径，用 `UIImagePickerControllerMediaURL` 键来获取。获取路径而不是直接获取视频，有一个非常大的好处，能够节省电量和内存。视频路径可以传递给MPMoviePlayerViewController，在用户屏幕上播放。获取视频的代码：

```
var videoPath = info[UIImagePickerControllerMediaURL as NSURL
```

MPMoviePlayerViewController这个类能够让开发视频播放的工作更简单，给MPMoviePlayerViewController一个视频路径，就能播放视频，也能在播放过程中做一些简单的操作。MPMoviePlayerViewController需要Media Player framework这个框架。

首先点击Project Navigator中的工程名称，显示工程的详细信息，然后滑到底部，找到Linked Frameworks and Libraries这部分，点击左下角的加号，在搜索框中输入MediaPlayer，选择MediaPlayer.framework，点击Add按钮。

这样MediaPlayer.framework文件就添加到你的Project Navigator中了，接着你需要打开view controller，然后把鼠标放到import UIKit下方，然后添加这行代码：

```
import MediaPlayer
```

这行代码会把MediaPlayer.framework类引入到当前代码中，MediaPlayer.framework中包括MPMoviePlayerViewController这个类。

接着，和创建其他的对象一样，创建一个MPMoviePlayerViewController，接着设置它的 `contentUIRL` 属性：

```
var videoPath = info[UIImagePickerControllerMediaURL as NSURL]
var myMoviePlayerViewController = MPMoviePlayerViewController()
myMoviePlayerViewController.moviePlayer.contentURL = videoPath
```

Presenting UIImagePickerController

展示新的view controller，我们使用 `presentViewController` 这个方法：

```
self.presentViewController(imagePicker, animated: true, completion: nil)
```

> UIImagePickerController | Page 241

Integrating with Social Networks （加入社交网络功能）

在很多手机应用中，把内容分享到社交网络上已经成为一个核心的特性，然而，把所有的社交网络都整合到App中会非常花费时间，苹果公司提供了Social framwork让开发分享功能更简单。Social framework中包括SLComposeViewController，SLComposeViewController这个类可以让用户把内容分享到Twitter和Facebook。SLComposeViewController使用用户在iPhone设置中填写的登录帐号密码，这就说明开发者不必自己写代码就可以分享到Facebook和Twitter了，用户可以分享文字、链接甚至是图片。

在使用SLComposeViewController之前，我们需要先把Social framework引入到工程当中。首先点击Project Navigator中的工程名称，显示工程的详细信息，然后滑到底部，找到Linked Frameworks and Libraries这部分，点击左下角的加号，在搜索框中输入Social，选择Socail.framework，点击Add按钮。这样Social.framework文件就添加到你的Project Navigator中了，接着你需要打开view controller，然后把鼠标放到import UIKit下方，然后添加这行代码：

```
import Social
```

Setting the Social Network （设置社交网络）

上面这行代码会把Social framework引入到view controller中，使SLComposeViewController可以在代码中使用了。在创建SLComposeViewController时需要提供serviceType，serviceType类型有两个选项： `SLServiceTypeFacebook` Facebook `SLServiceTypeTwitter` Twitter

首先要核实需要的service是否可用，使用 `isAvailableForServiceType(_: )` 来检查service类型在当前设备是否可用：

```
if (SLComposeViewController.isAvailableForServiceType(SLServiceTypeFacebook)) {
     //Facebook available
}
```

我们可以使用_(forServiceType:)方法来创建SLComposeViewController，有一个forServiceType参数：

```
if (SLComposeViewController.isAvailableForServiceType(SLServiceTypeFacebook)) {
       //Facebook available
      var myComposeViewController = SLComposeViewController(forServiceType: SLServiceType
}
```

> Page 242 | Chapter 9: Camera, Photos, and Social Networks

Setting the Initial Text （设置默认文案）

你还可以给SLComposeViewController设置默认文案，这段默认文案会出现分享界面，提前给用户写好，用户可以直接就分享出去，当然也可以删除或者修改文案。我们用setInitialText(_:)方法来创建默认文案：

```
var myComposeViewController = SLComposeViewController(forServiceType: SLServiceTypeFacebo
myComposeViewController.setInitialText("I love this app!")
```

Adding Images （增加分享图片）

SLComposeViewController也支持图片分享，使用addImage(_:)方法，这个方法有一个UIImage参数：

```
myComposeViewController.addImage(myImage)
```

Adding URLs

如果不能分享链接那么分享就没有意义了，SLComposeViewController当然支持分享URL，使用addURL(_:)方法，接收NSURL作为参数。NSURL和字符串非常相似，专门为UIL和文件路径使用。例如：

```
var myURL = NSURL(string: "http://www.google.com")
myComposeViewController.addURL(myURL)
```

Presenting SLComposeViewController

最后，SLComposeViewController创建并且设置完成后，需要把SLComposeViewController呈现给用户了，我们使用modally present这个呈现方式，使用`presentViewController(_:,animated: completion: )`方法，例如：

```
self.presentViewController(myComposeViewController,animated: true, completion: nil)
```

这样SLComposeViewController就会展示在用户面前，用户分享内容。如果用户还没有登录社交网络，iOS会引导他到设置中的登录界面完成操作。

现在是时候把知识应用到实践中去了，来创建一个自拍照App吧。

> Integrating with Social Networks | Page 243

第九章练习 A Selfie App - 开发一个自拍App吧

在本书的剩下的几章中，你将学习开发Selfie App然后提交到App Store。Selfie（自拍）App能够让用户拍照然后分享到Facebook（见图9-2），仅能在iPhone中使用，同时使用这章节学到的UIImagePickerController的相关知识点。



打开Xcode，顶部菜单栏选择File -> New Project。选择Single View Application，点击Next。

> Page 244 | Chapter 9: Camera, Photos, and Social Network

Name一栏输入Selfie，Organization Name和Organization Identifier设置为你的名字不要有空格（见图9-3）。语言是Swift，Device选择iPhone，点击Next，保存文件。



屏幕显示工程信息信息，在Device Orientation区域，不勾选Landscape Left和Landscape Right选项（见图9-4）。



> Exercise: A Selfie App | Page 245

滑倒底部找到Linked Frameworks and Libraries区域（见图9-5），点击左下角的加号，在搜索框中输入Social，选择Social.framework文件，点击Add。Social.framework文件出现在Project Navigator中了，把它拖到Supporting Files文件夹中。



> Page 246 | Chapter 9: Camera, Photos, and Social Network

打开Main.storyboard文件（见图9-6）。从Object Library拖拽一个Image View，放到界面的中间，把Image View的四个边界拖拽到和界面一样大小，在Attribute Inspector中，把mode改为Aspect Fill。



选中Image View，从顶部菜单栏选择Editor -> Pin -> Leading Space to Superview；再次选中Image View，从顶部菜单栏选择Editor -> Pin -> Trailing Space to Superview；再次选中Image View，从顶部菜单栏选择Editor -> Pin -> Top Space to Superview；再次选中Image View，从顶部菜单栏选择Editor -> Pin -> Bottom Space to Superview。这样Image View就能够适配任何尺寸的设备了（见图9-7）。



> Exercise: A Selfie App | Page 247

接下来你要添加一个navigation bar和两个按钮（见图9-8）。点击选中当前界面上的黄色圆圈，点击顶部菜单栏的Editor -> Embed In -> Navigation Controller。Navigator Controller出现在界面上了，双击navigation bar中间部分，输入Selfie。



> Page 248 | Chapter 9: Camera, Photos, and Social Network

从Object Library中拖拽两个Bar Button Item到界面上，放到navigation bar的两侧，双击左侧Bar Button Item，命名为Take Selfie。双击右侧Bar Button Item，命名为Share（见图9-9）。



现在界面上的控件已经放置完毕了，需要把控件和ViewController.swift文件建立连接。打开Assistant Editor，隐藏Inspector和Document Outline，删除去掉viewDidLoad和didReceiveMemoryWarning方法。

接着按住Control键，点击Image View拖拽到ViewController.swift文件中，放到 class ViewController: UIViewController（见图9-10）的下方，然后松开鼠标，出现弹出框（见图9-11）。





> Exercise: A Selfie App | Page 249

Connection选择Outlet，Name一栏输入myImageView，点击Connect。

按住Control键，把Take Selfie按钮拖拽到ViewController.swift文件中，放到 class ViewController: UIViewController（见图9-12）的下方，然后松开鼠标，出现弹出框（见图9-13）。





> Page 250 | Chapter 9: Camera, Photos, and Social Network

Connection类型选择Action，Name一栏中输入selfieTapped，点击Connect。selfieTapped自动出现在ViewController.swift文件中。

按住Control键，把Share按钮拖拽到ViewController.swift文件中，放到 class ViewController: UIViewController（见图9-14）的下方，然后松开鼠标，出现弹出框（见图9-15）。





> Exercise: A Selfie App | Page 251

Connection类型选择Action，Name一栏中输入shareTapped，点击Connect。shareTapped自动出现在ViewController.swift文件中。

在selfieTapped方法中添加下列代码：

```
@IBAction func selfieTapped(sender: AnyObject){
    var imagePicker = UIImagePickerController()
    imagePicker.delegate = self

    self.presentViewController(imagePicker, animated: true, completion: nil)

}
```

> Page 252 | Chapter 9: Camera, Photos, and Social Network

第一行代码创建了UIImagePickerController并赋值给了变量imagePicker，第二行代码设定imagePicker的委托属性是当前的view controller，也就是.self。最后一行代码呈现imagePicker，呈现方式是modal，从屏幕下方出现，动画效果为真true。

刚刚写完的这方法出现了错误警告，imagePicker.delegate这行有问题，点击红点，显示：

Type “ViewController” does not conform to protocol ‘UIImagePickerControllerDelegate

这是Xcode让你知道ViewController实例必须要遵守UIImageViewControllerDelegate和UINavigationControllerDelegate两个协议。修改为下列代码：

```
class ViewController: UIViewController, UIImagePickerControllerDelegate, UINavigationCont
```

按Command+B编译程序。Command+B编译程序能够编辑所有的代码但是不启动模拟器，在修复bug时，比较有帮助。

把鼠标光标放到imagePicker.delegate = self这行代码下方，增加下列代码：

```
if UIImagePickerController.isSourceTypeAvailable(.Camera) {

}  else  {

}
```

if语句检查当前设备的相机是否可用，如果可用，执行上半部分的代码，如果不可用，执行下半部分的代码。把if语句补充完整：

```
if UIImagePickerController.isSourceTypeAvailable(.Camera) {
    imagePicker.sourceType = .Camera

    if (UIImagePickerController.isCameraDeviceAvailable(.Front)) {
        imagePicker.cameraDevice = .Front
    } else {
        imagePicker.cameraDevice = .Rear
    }

}
```

上半部分中第一行代码是把imagePicker的sourceType属性设置为.Camera，第二行开始是一个if语句检查前置摄像头是否可用，如果可用，就把imagePicker.cameraDevice设置为.Front，这样默认摄像头就是前置摄像头了。如果不可用，使用后置摄像头。

> Exercise: A Selfie App | Page 253

完成if语句isSourceTypeAvailable下半部分的代码：

```
 } else {
          imagePicker.sourceType = .PhotoLibrary
    }
```

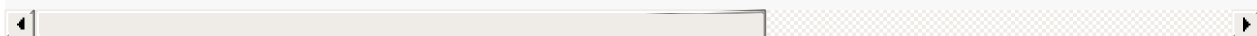
当没有可用的摄像头时，下半部分的代码就会执行，这时PhotoLibrary就会出现。（见图9-16）



接下来添加didFinishPickingImage方法：

```
func imagePickerController(picker: UIImagePickerController!,didFinishPickingImage image:

}
```

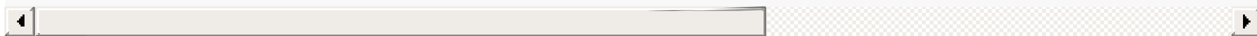


当用户选择了一张照片或者拍摄照片后，这个方法就会被调用。didFinishPickingImage方法有两个参数，第一个是UIImagePickerController，第二个参数是名为Image的UIimage，image参数存储用户选择或拍摄的那张照片。

修改添加下列代码：

```
func imagePickerController(picker: UIImagePickerController!,didFinishPickingImage image:

    myImageView.image = image
    self.dismissViewControllerAnimated(true, completion: nil)
}
```



> Page 254 | Chapter 9: Camera, Photos, and Social Network

第一行代码是把image参数值赋值给myImageView.image，这样myImageView就会展示image参数图片了。第二行代码是隐藏imagePicker。

把鼠标光标放到import UIKit下方，然后添加代码：

```
import Social
```

这行代码把Social.framework引入到ViewController.swift文件中，把鼠标光标放到shareTapped方法中，当用户点击Share按钮时，这个方法就被调用。在这个方法中需要完成两件事情：添加图片，显示Facebook的分享对话框。

在shareTapped():中添加下列代码：

```
var social = SLComposeViewController(forServiceType: SLServiceTypeFacebook)
social.addImage(myImageView.image)

self.presentViewController(social, animated: true, completion: nil)
```

第一行代码创建了SLComposeViewController，设置service属性为Facebook。接着第二行代码把myImageView中的图片添加到Facebook分享框中，最后第三行代码，显示SLComposeViewController。

点击Play按钮（Command+R），App启动后，选中图片后点击Share，回车耍嘛一个提示框，显示iOS 模拟器没有Facebook帐号，需要去设置中添加Facebook帐号。一旦设置完成后，停止运行App，再次Command+R启动App（见图9-17）。



> Exercise: A Selfie App | Page 255

如果App没有按照你想要的结果运行，或者程序有了错误或警告，不要太担心，学习的最佳方式就是试错，熟能生巧，到我们的网站上下载示例代码，对比一下代码，多试几次，直到搞定这个程序为止。

> Page 256 | Chapter 9: Camera, Photos, and Social Network

第十章：在真机上运行

在这一章节，你将学会如何在真机上运行你的App，你将学会如何创建证书、注册你的测试设备，设置你的App ID和provisioning profiles。这章节将介绍和练习部分合二为一，所以在看这一章节时，要确保打开Xcode，边看边操作。

真机上测试App需要注册开发者账户（Apple Developer account）（现在苹果已经允许开发者没有帐号也能在真机上运行了，作者写这本书的时候，苹果的政策没有开放。）,将App发布到App Store上也需要这个账户。Apple Developer Program为开发者提供了最新版本的iOS和OSX，注册开发者帐号可以观看WWDC中的视频。WWDC全称Worldwide Developer Conference，每年在San Francisco举行。苹果公司会在WWDC上发布新的产品和软件，注册开发者还能获得专门的技术支持，每年需要缴纳99美元，这将是你一笔划算的投资。

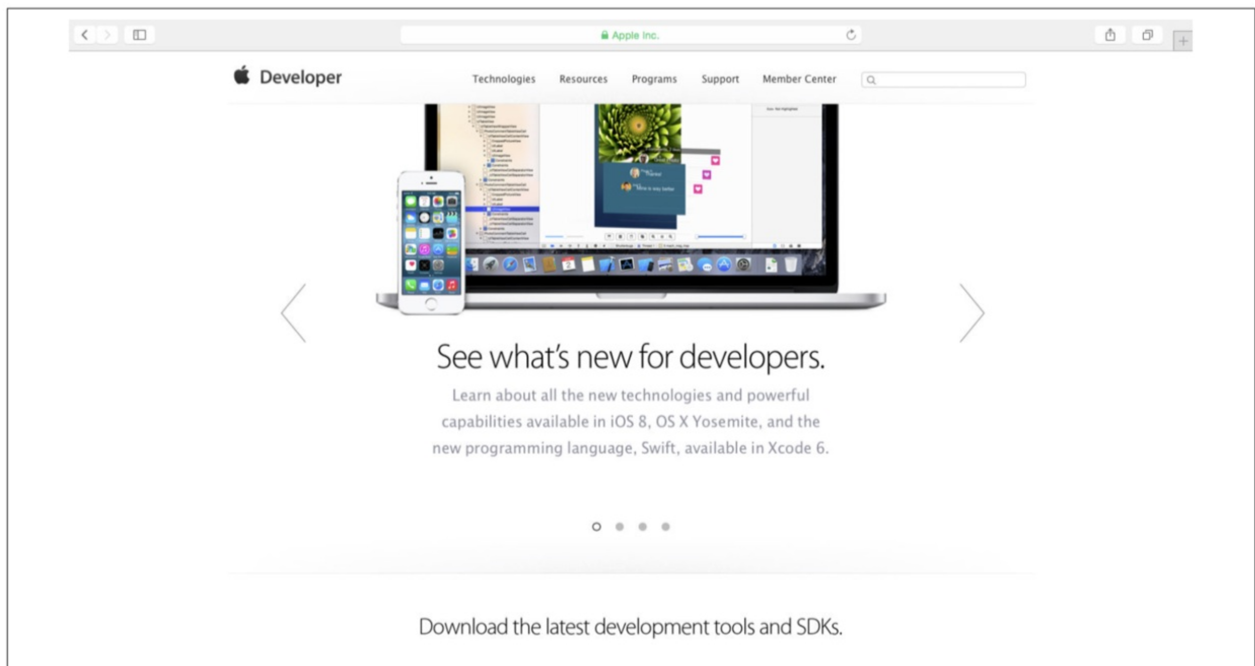
> 
> 此章节接下来的部分需要注册开发者账户，你可以在http://developer.apple.com/programs/ios.上注册账户 如果你是在校学生，苹果提供了iOS Developer University Program免费项目，更多细节可以在这里找到http://developer.apple.com/programs/ios/university

可以用你的名字或者公司名称来注册开发者账户，注册在你个人名下就是个人开发者账户，注册在公司名下需要公司的邓白氏编码和可证明公司的合法文件。邓白氏编码是你公司独一无二的编码，你可以在https://developer.apple.com/support/ios/ D-U-N-S.php这个网站上了解更多的信息。在大多数情况下，注册成个人开发者会更简单一些，当你需要的时候，可以把个人开发者账户转换成公司账户。更多信息，请见https:// developer.apple.com/programs/。

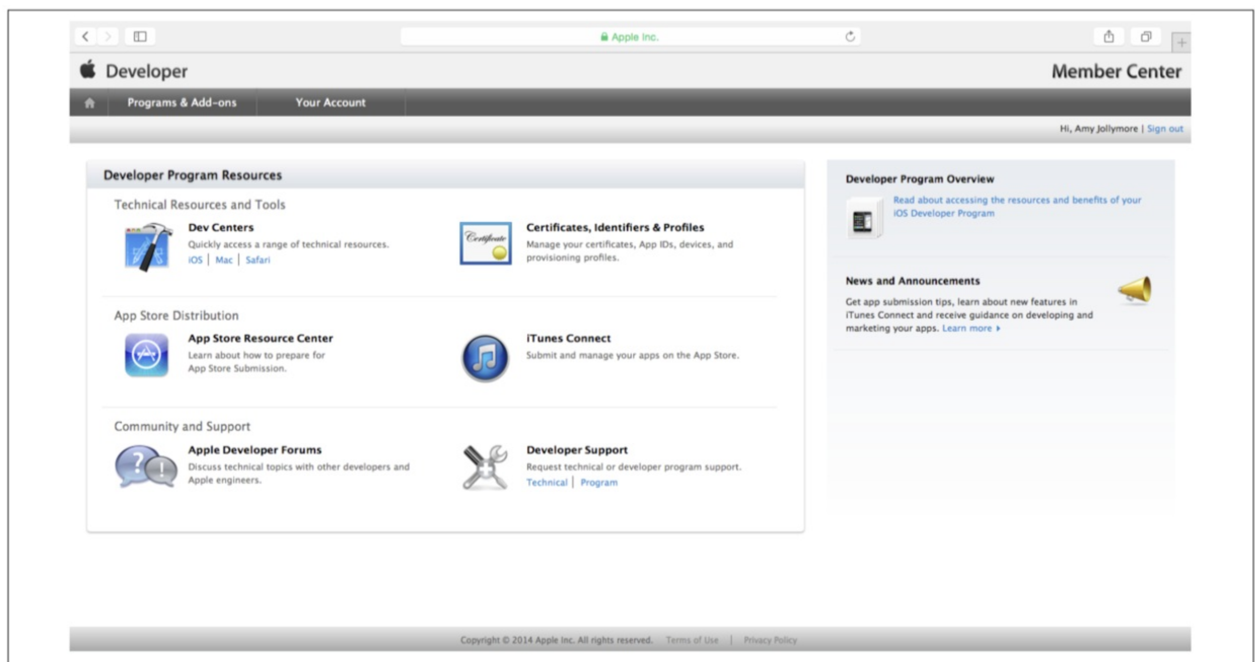
> Page 257

完成注册后，就可以进入Member Center。Member Center中提供了苹果开发者需要的工具、更新，在https://developer.apple.com/member center/注册登录Member Center。Member Center中有部分叫做*Certifications, Identifiers, and Pro-files*，这部分就是*Provisioning Portal*，控制管理在你开发过程中需要使用的设备和profiles。

用浏览器打开http://developer.apple.com（见图10-1）



点击右上角的Member Center，输入你的苹果开发者帐号密码，然后登录（见图10-2）。



> Page 258 | Chapter 10: Running on a Device

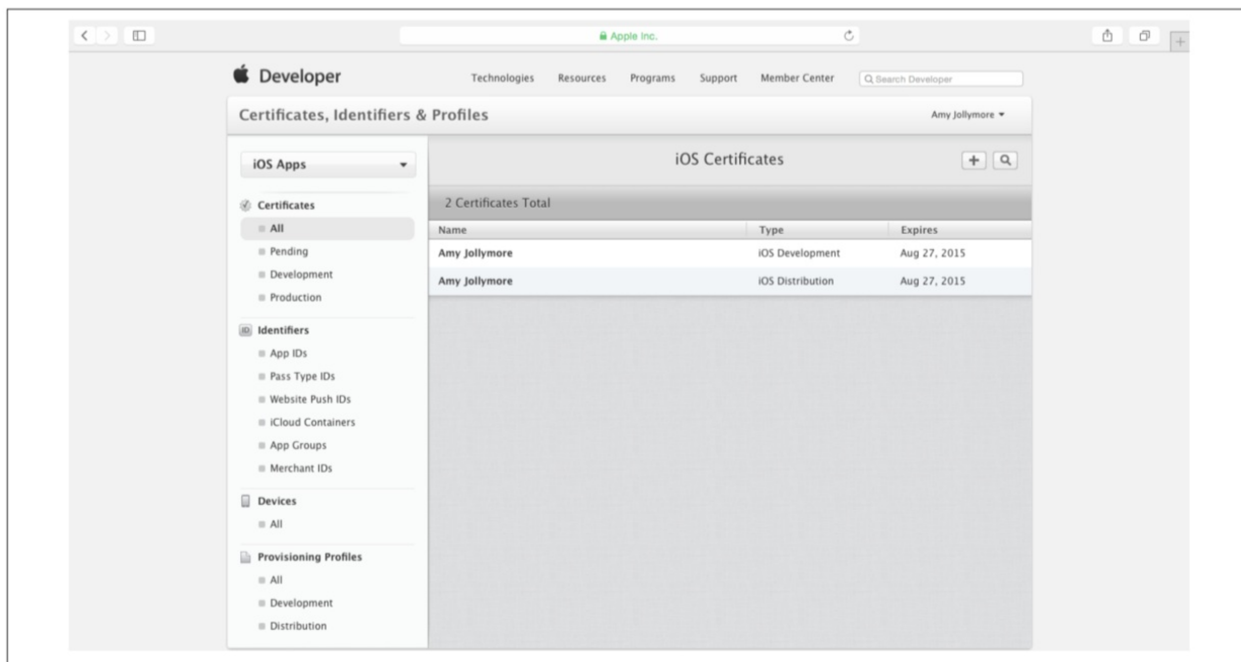
点击“Certifications, Identifiers and Profiles”这个链接，接着点击“Certificates”链接。

Provisioning Portal有四部分:Certificates、Identifiers、Devices、Profiles，把这页加入收藏夹，因为我们以后会经常用到这个页面。

Certificates

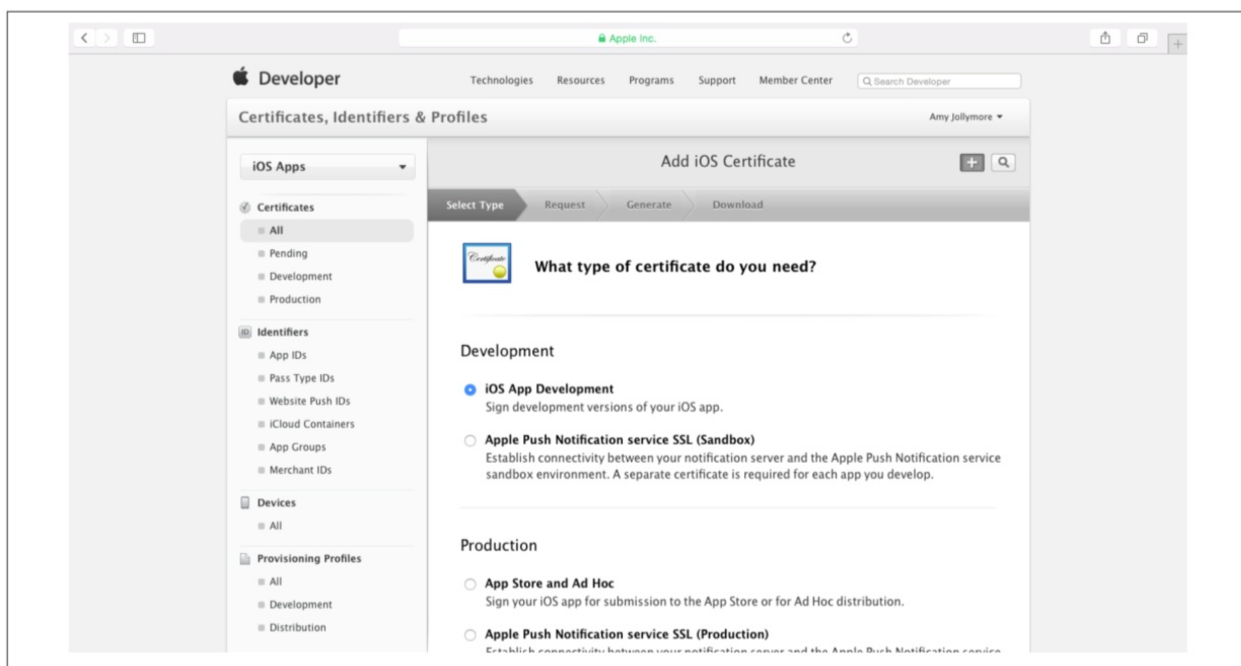
第一部分是Certificates.在你运行App时，Certificates用来确认你的身份。唯一的certificate就像是你的签名，这样能够限制其他人在未经你允许的情况下向App Store提交App。

打开Certificate部分后，你会看到一个银色入口（见图10-3）。左边是一排选项，Pending表示你目前正在使用的一些certificate，Development certificates表示用于本地电脑上的证书，Production certificates是用在App Store上的。



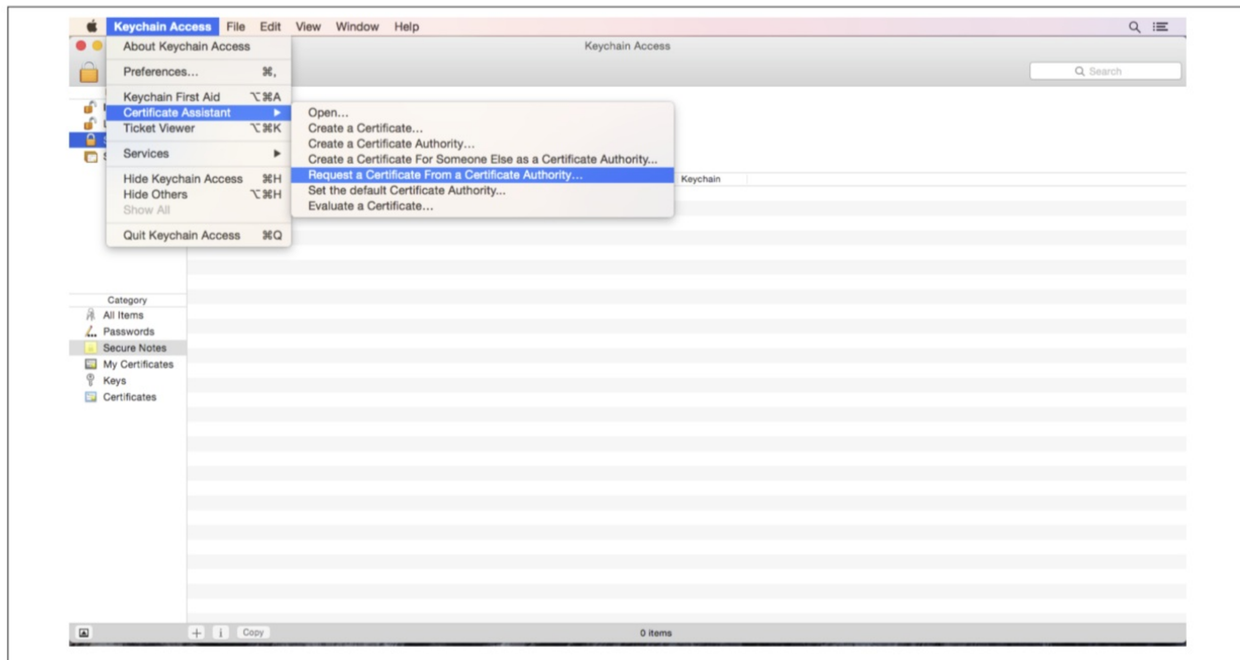
Certificates | Page 259

点击右上角的加号按钮，选择iOS App Development，然后点击Continue继续。接着会出现向导，让你想创建的证书的类型（见图10-4）。选择iOS App Development box，点击Continue。向导接下来会让你创建CSR（Certificate Signing Request），CSR是创建Certificate的一个条件。

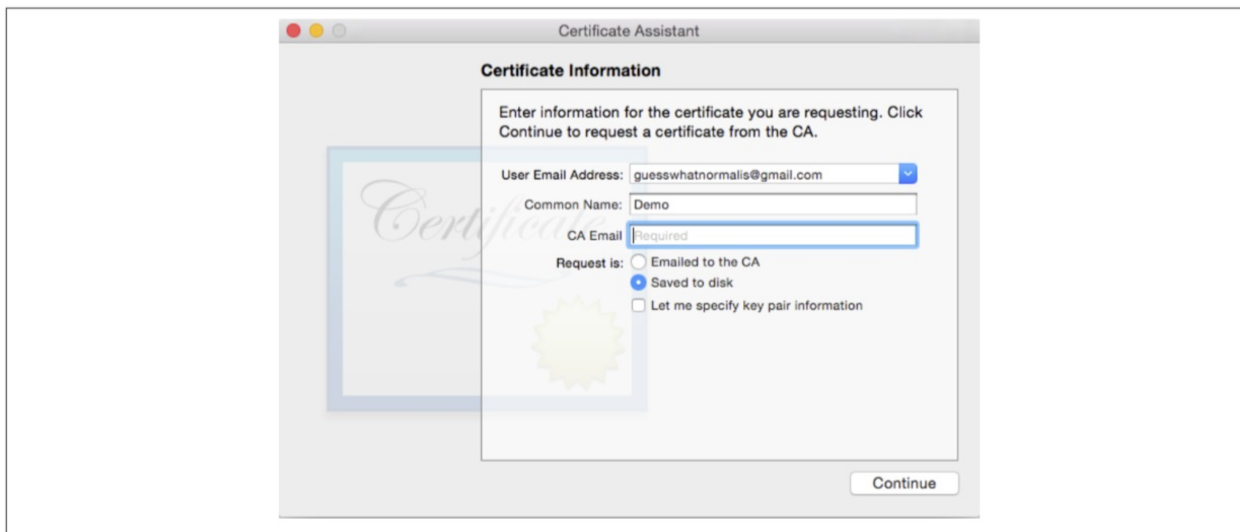


Page 260 | Chapter 10: Running on a Device

想要创建CSR，我们首先要打开你Mac中的Keychain Access这个应用程序。点击Mac屏幕右上角的Spotlight搜索图标，输入**keychain Access**，然后点击搜索出来的第一个选项。Keychain Access启动后会列出你当前电脑上所有的certificates。点击顶部菜单栏中的Keychain Access，然后选择Certificate Assistant --> Request a Certificate From a Certificate Authority（见图10-5）。

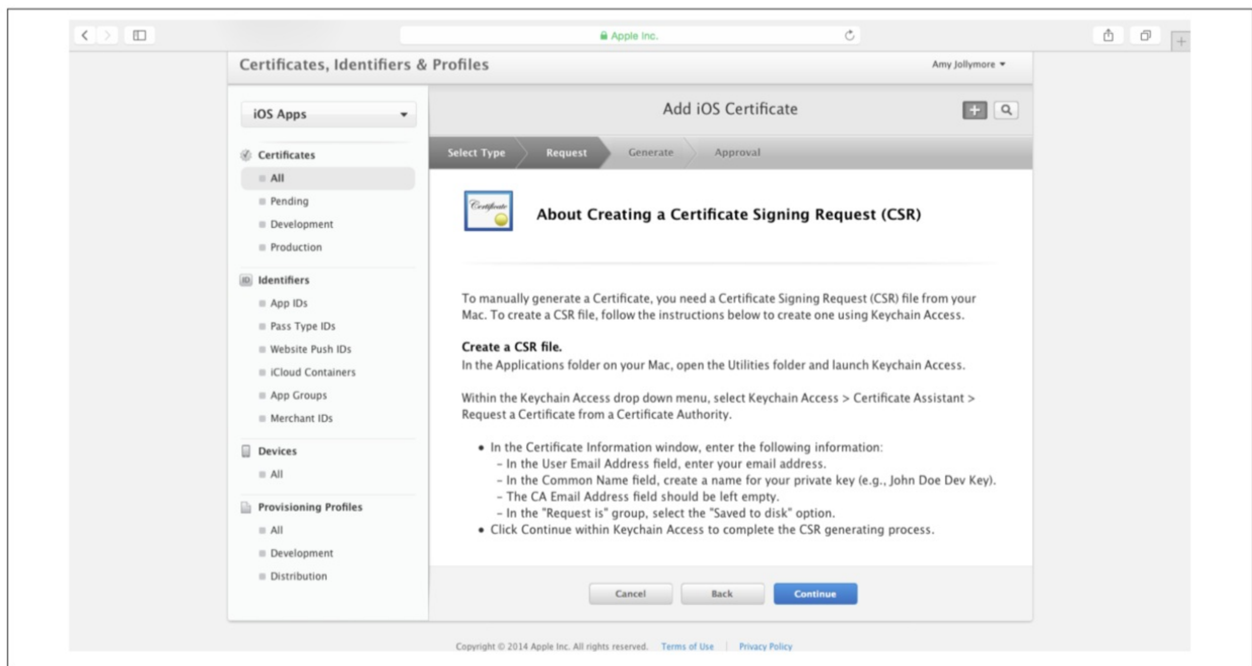


接着屏幕上会出现图10-6的向导。填写email和common name，CA Email address这一栏空着，然后勾选“Saved to disk”，点击Continue。选择存放证书的文件夹，然后保存。现在回到浏览器继续Provisioning Portal。

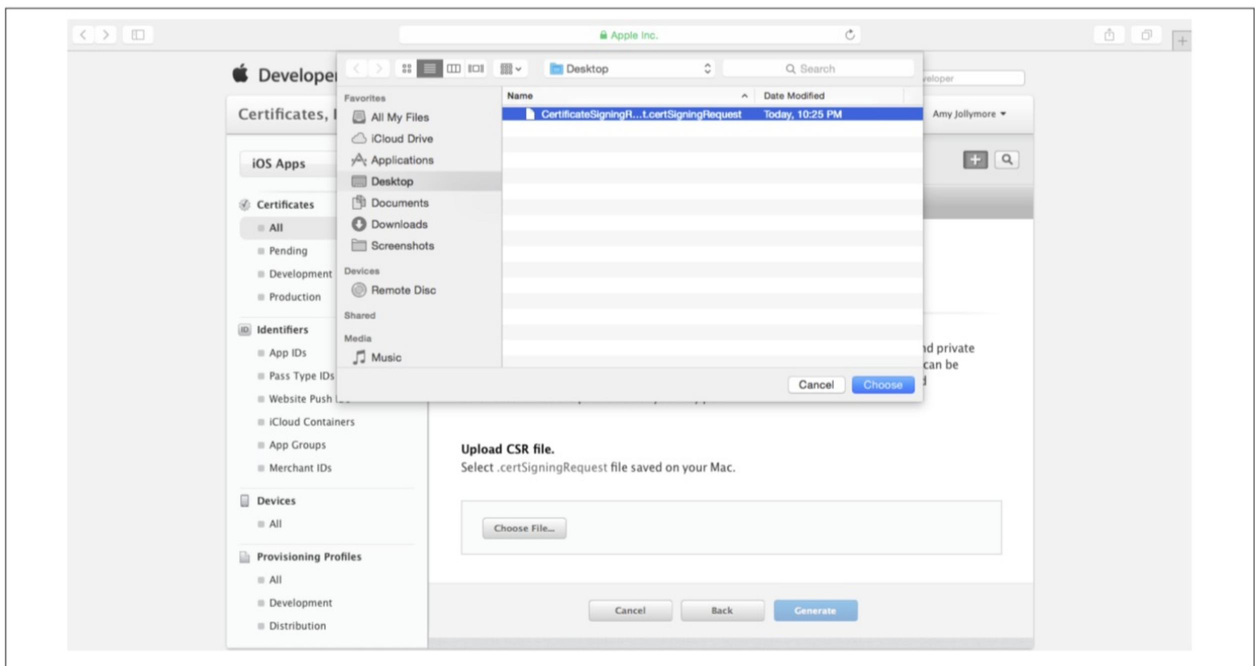


Certificates | Page 261

创建好CSR后，点击浏览器中Continue按钮（见图10-7）。

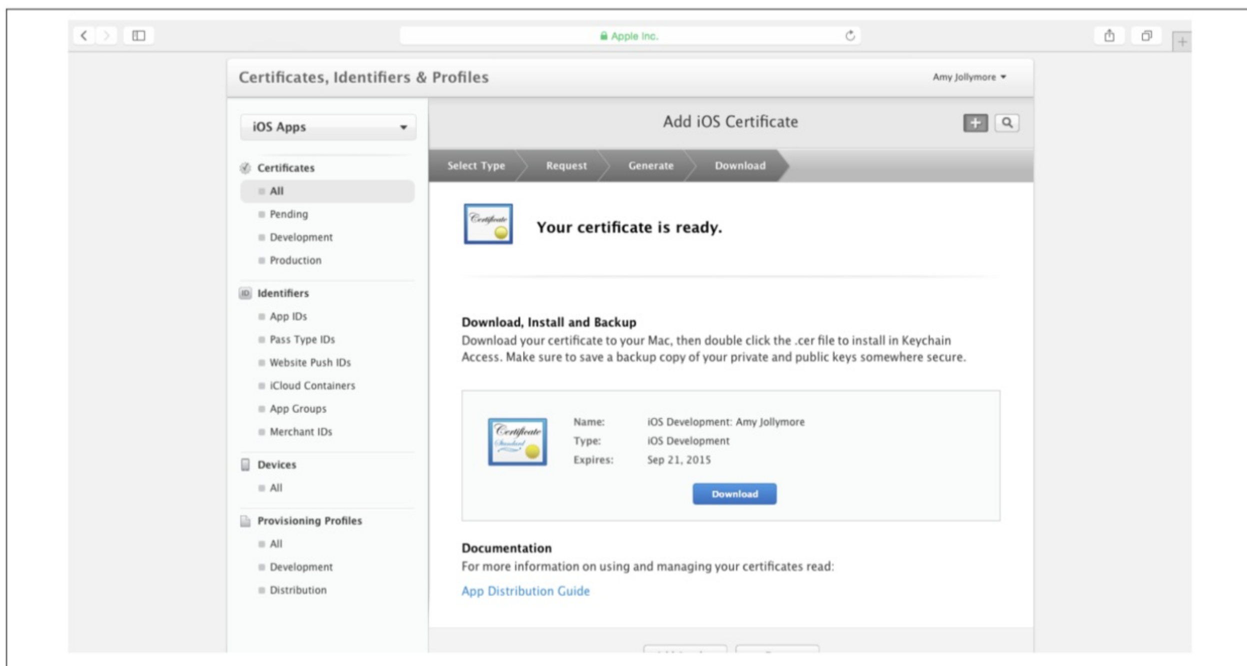


接下来需要你上传CSR文件。点击Choose File按钮，找到*.certSigningRequest*文件（见图10-8）。



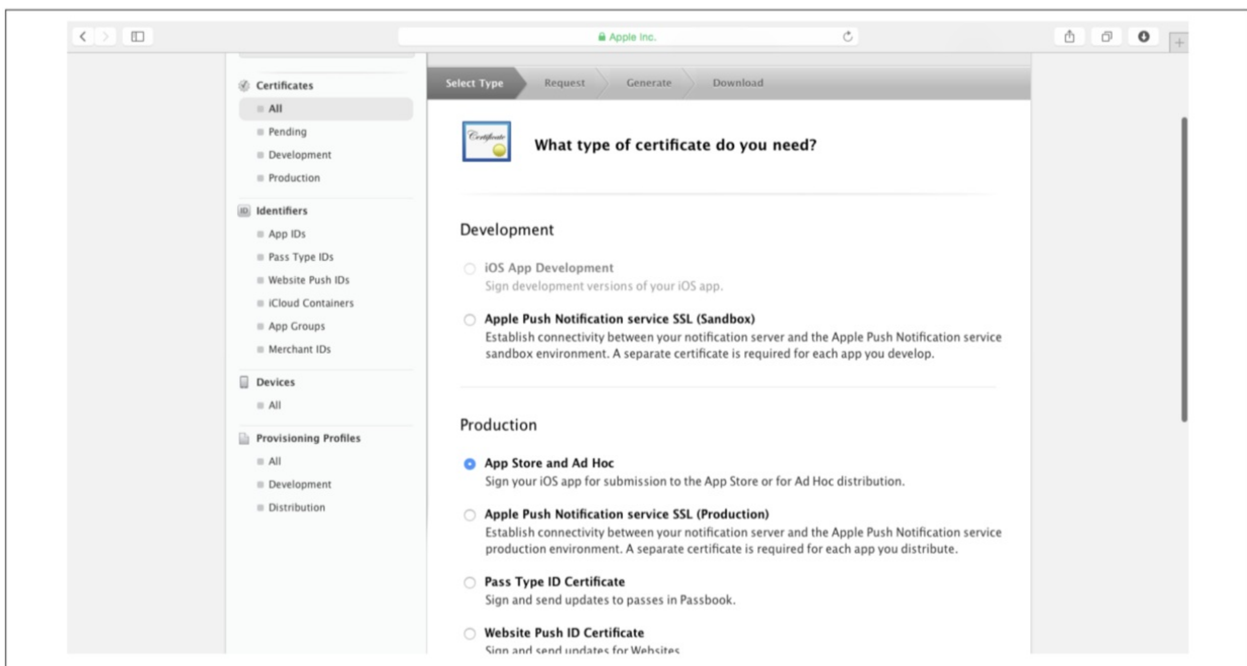
Page 262 | Chapter 10: Running on a Device

接下来点击Generate，向导开始创建证书。证书创建好后，会出现一个Download按钮（见图10-9）。点击Download下载证书，开发Downloads文件夹，双击刚刚下载的*ios_development.cer*文件，这样就把文件添加到你的keychain中了。

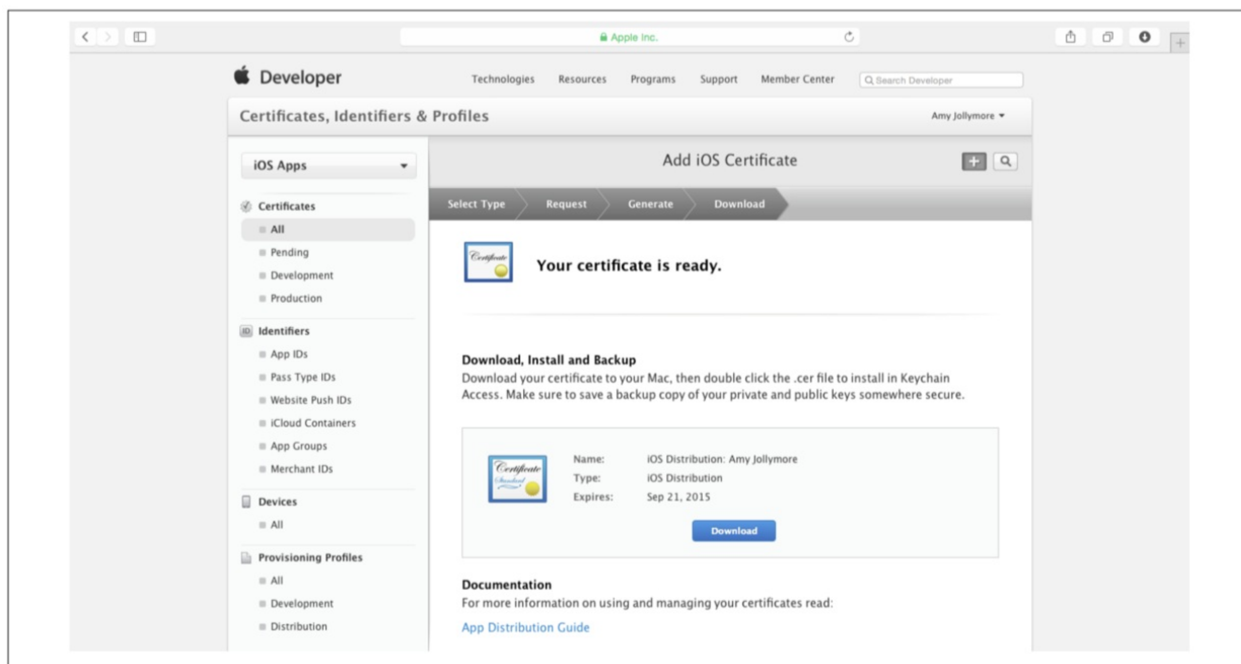


Certificates | Page 263

再次打开浏览器，点击左边的All选项。用于App Store的第二个证书也需要创建。再次点击右上角的加号按钮，选择“App Store and Ad Hoc”点击Continue（见图10-10）。CSR创建界面这时会弹出来，这一次，你用上次已经创建好的CSR就可以了。点击Continue。



点击Choose File按钮选择之前创建好的*.certSigningRequest*，点击Generate按钮。出现Download下载按钮，点击Download下载新的证书（见图10-11）。双击*ios_distribution.cer*文件即可添加到keychain中。然后再次打开浏览器。

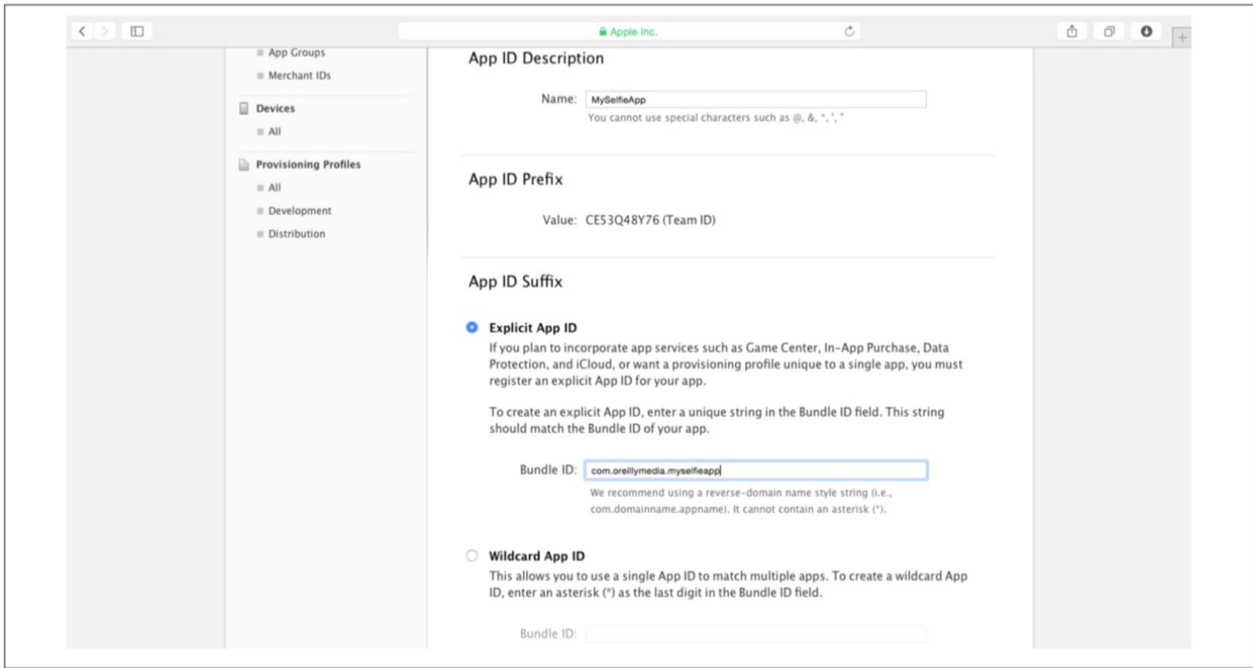


Page 264 | Chapter 10: Running on a Device

Identifiers

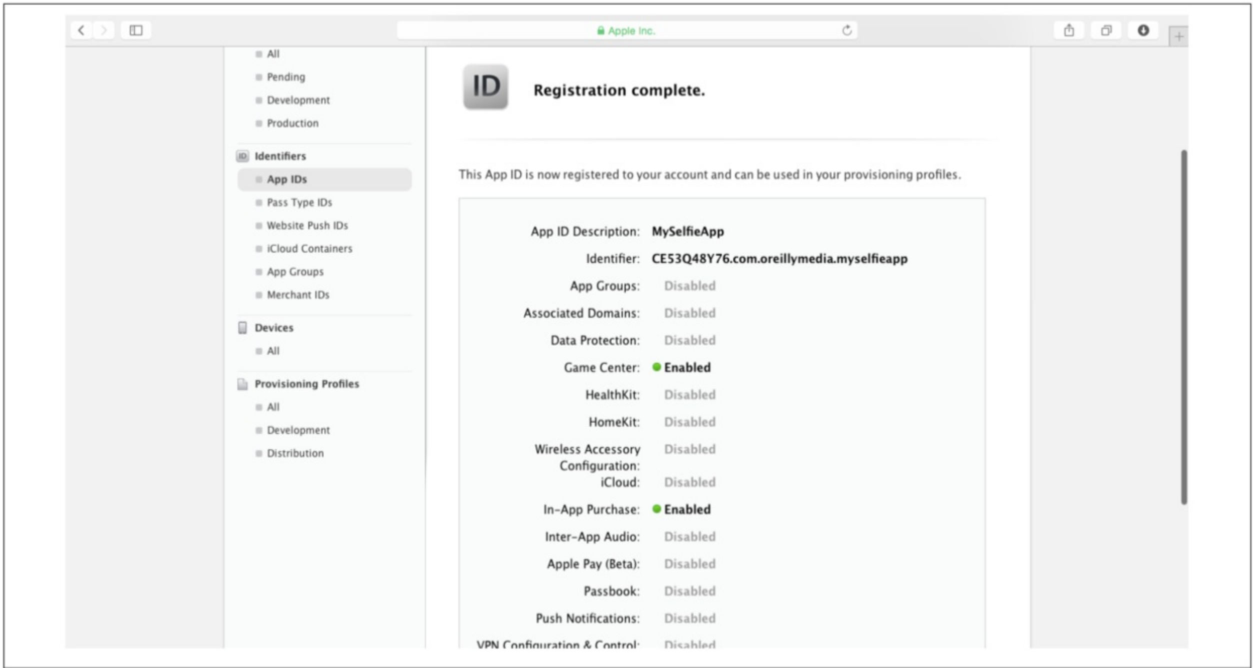
Provisioning Portal的第二部分是Identifiers，在Certificates下方。Identifiers是用来管理App的ID的。有点类似美国人的社保号，每个App都有自己独一无二的App ID。

要创建一个App ID，我们首先要左边的侧边栏中点击App ID，然后点击右上角的加号按钮，出现创建App ID的界面（见图10-12）。Name一栏中输入App ID Selfie，不要使用下方列出的特殊符号。App ID Prefix一栏空着，勾选Explicit App ID。Bundle ID这部分使用部分倒序的方式书写，就像是在浏览器地址栏中输入网址，只不过顺序是相反的。不用使用图片中的Bundle ID，例子中的Bundle ID是无效的。输入你自己的Bundle ID，确保使用以下格式：*com.domain.app*（例如：*com.johnsmith.selfieapp*）。



Identifiers | Page 265

Bundle ID会进入Xcode中，当你向App Store提交App时，会核对Bundle ID。App Services，你可以为你的应用添加格外的服务。点击Continue。需要你再次确认App ID，如果各项都正确点击Submit。这样，App ID和Bundle ID都创建完成了（见图10-13）。



Page 266 | Chapter 10: Running on a Device

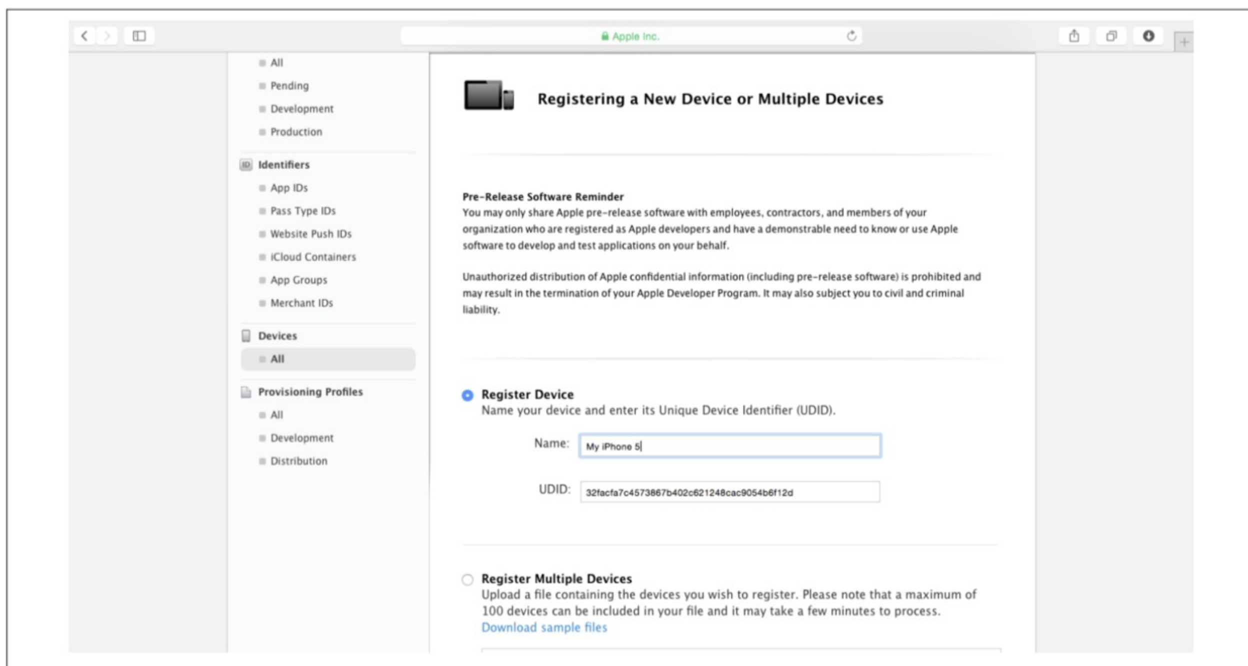
Devices

Provisioning Portal的第三部分是Devices。不能随笔拿个iOS设备就来真机测试你的应用，测试机需要注册。开发者可以每年注册最多100台测试设备，如果这个设备被移除了，就再也不能添加回来了。Devices部分包括了所有可以使用开发者账户的设备，我们使用一串唯一的串码UDID来关联设备。

点击Devices下方的All，然后点击右上角的加号按钮。

我们需要获取设备的UDID（见图10-14），获取UDID的方法是，先把设备链接到Mac上，运行Xcode，设备连接上后，打开Xcode，选择顶部菜单Window --> Devices，出现设备窗口，从左侧栏中选择iOS设备，这时iOS设备上会弹出是否信任这台电脑（Trust This Computer）的提示，点击信任（Trust）。UDID就会在Identifier后面，复制UDID，然后打开浏览器。

把UDID粘贴到输入框中，给这个设备命名（见图10-14），点击Continue。



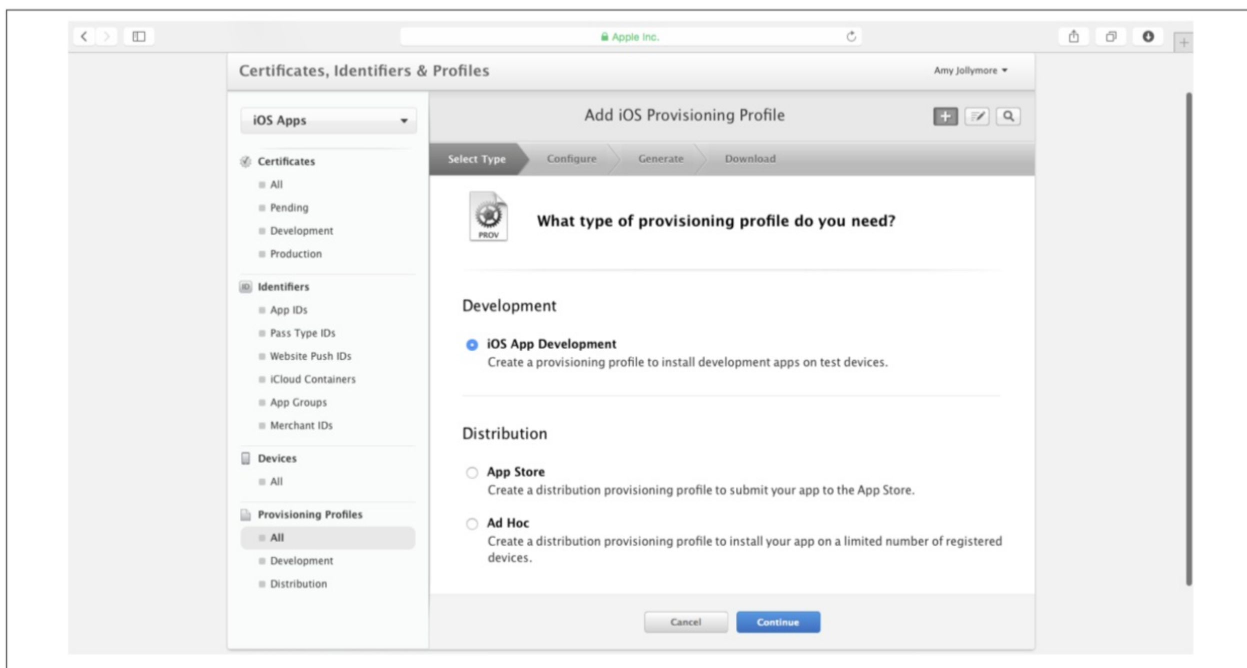
> Devices | Page 267

Profiles

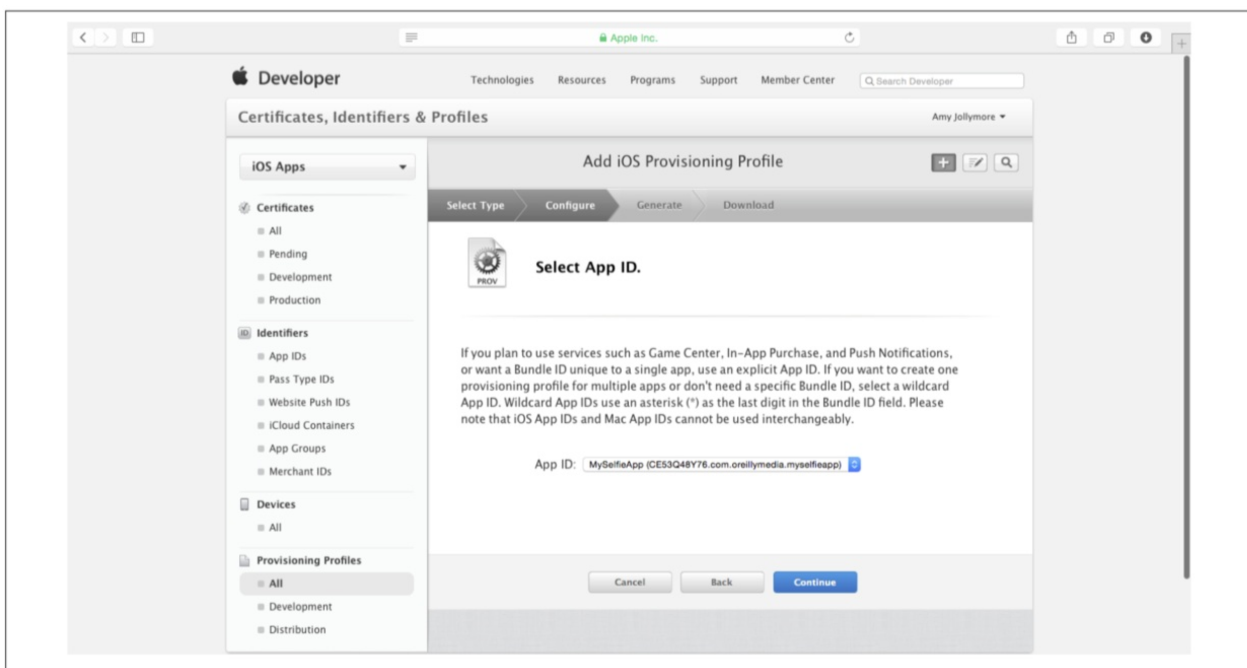
一旦你的设备已经激活了开发者模式且添加到了Devices中，你就可以创建provisioning profile了。*provisioning file*列举出哪些设备可以运行你的App。把provisioning profile添加到Xcode，然后检查此设备是否授权运行App，provisioning profile用于本地开发和App Store。然而，App Store的provisioning profile不限于具体的设备。

点击Provisioning Profiles下方的All，接着点击右上角的加号按钮。

选择iOS App Development，点击Continue（见图10-15）。Development用来在你电脑上运行App，Distribution是预留你的App提交到应用市场或者群体测试。

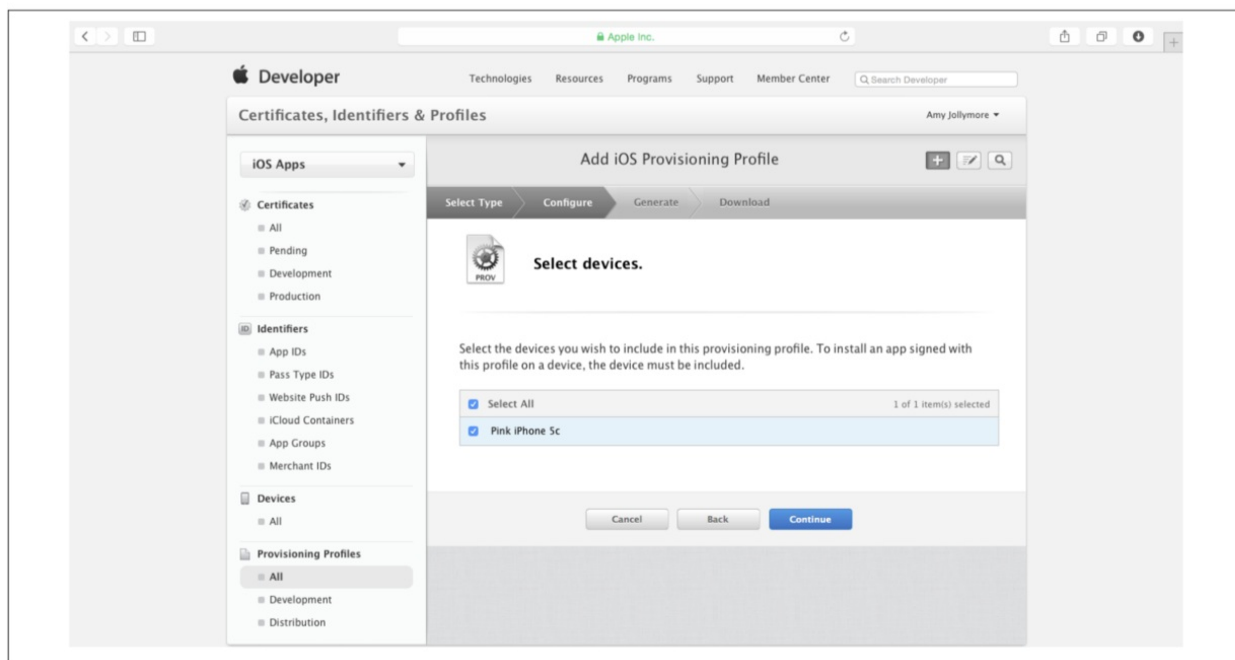


接下来，从下拉菜单中选择你的应用的 App ID，点击Continue（见图10-16）。选择你要使用此provisioning profile的证书，大部分情况下，你只需要一个可用证书即可。选择你的证书然后点击Continue。

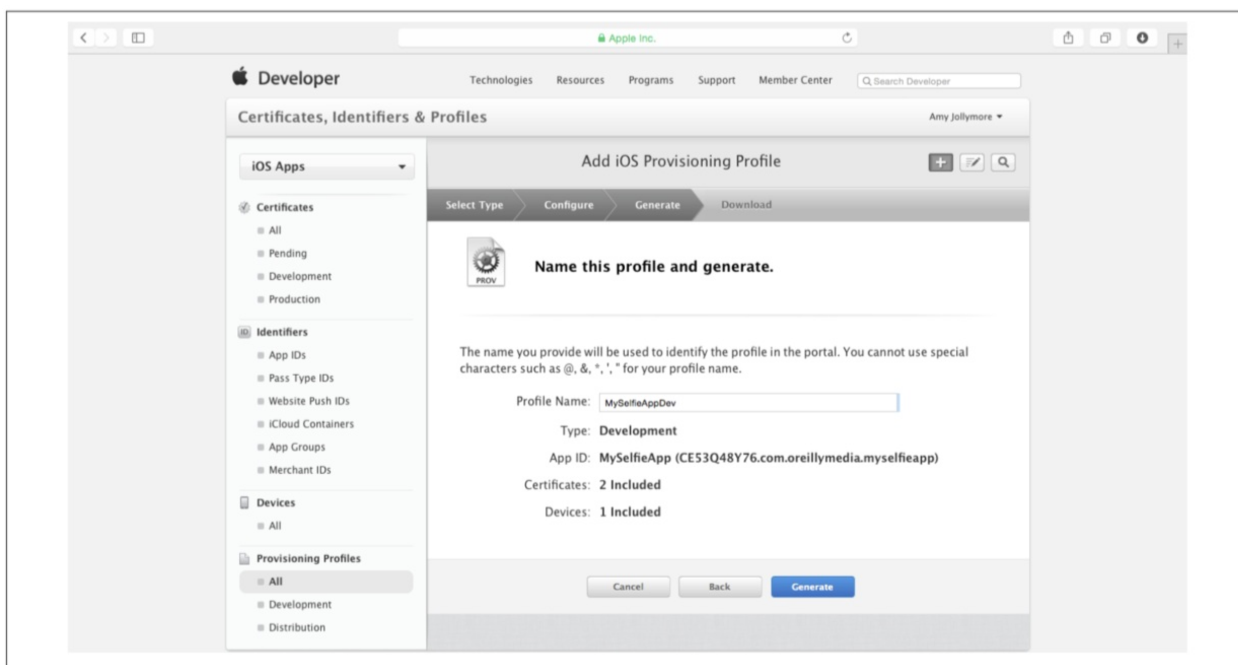


Page 268 | Chapter 10: Running on a Device

接下来，会出现一个你账户下授权设备清单（见图10-17）。选择你想要测试应用的设备。



最后，使用*projectNameDev*或*projectNameAppStore*格式来命名profile name（见图10-18）（例如：MySelfieAppDev）。使用这种格式命名，可以在Xcode中更容易找到provisioning profile。点击Generate，然后下载profile。

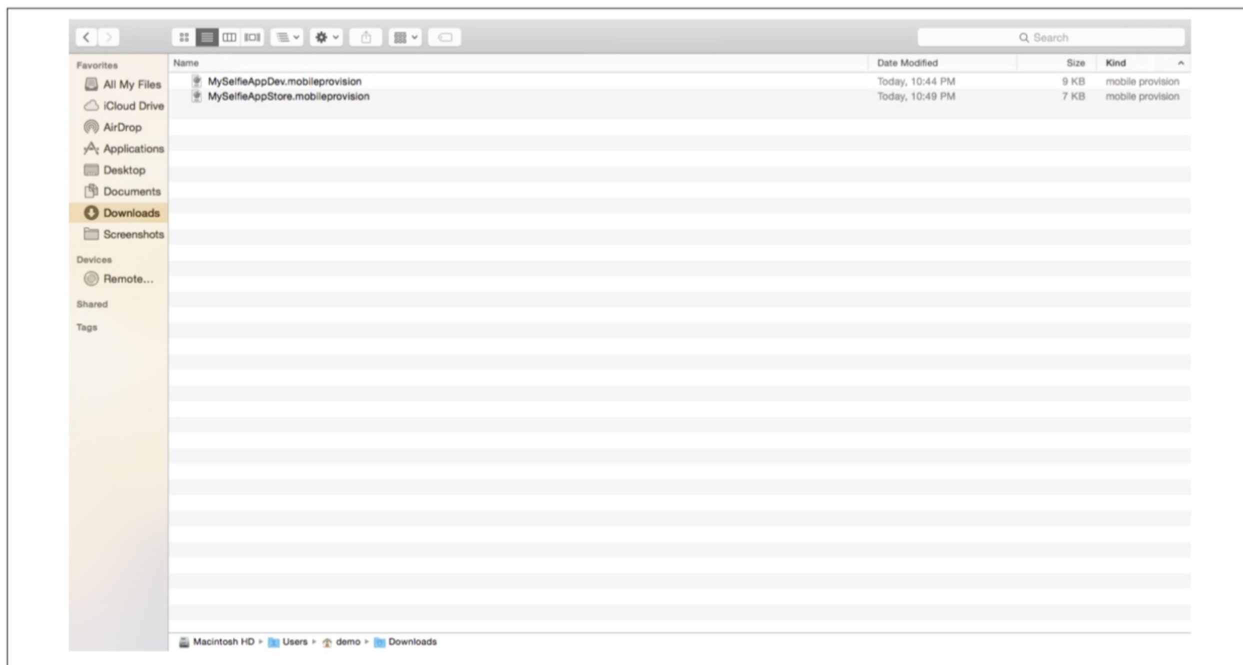


Profiles | Page 269

打开浏览器，点击Provisioning Profile下方的All，接着点击右上角的加号按钮。

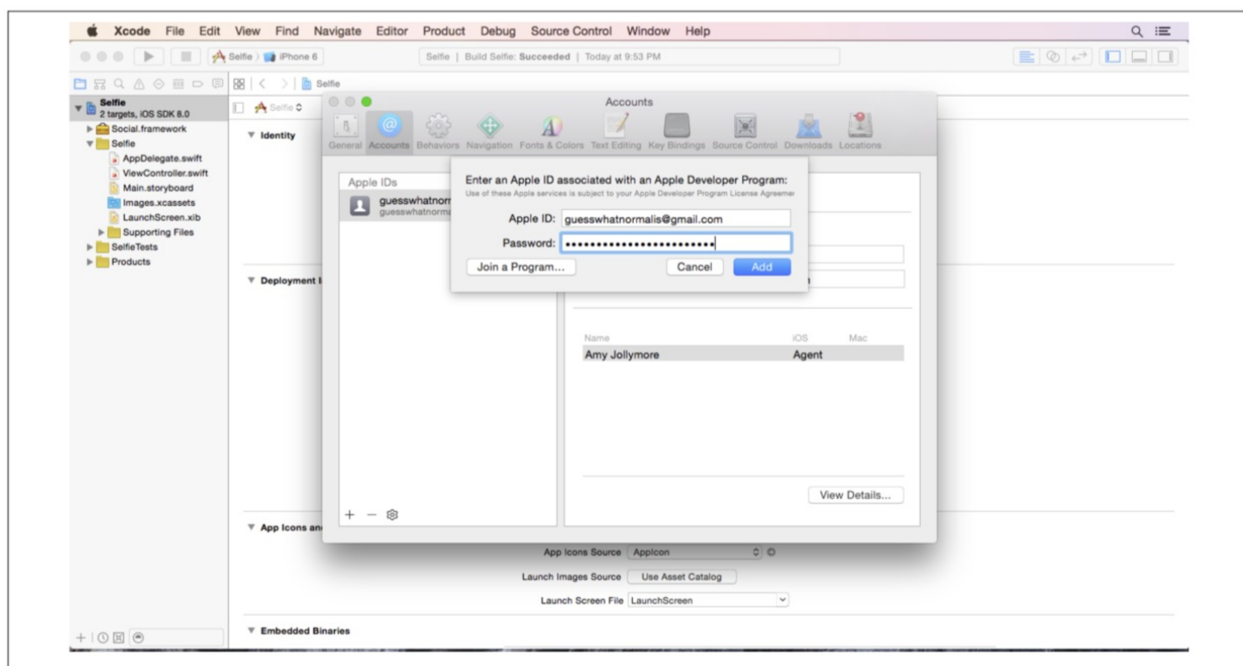
选择App Store点击Continue。从下拉菜单中选择App的App ID，点击Continue。选择证书，点击Continue。命名profile（例如：MySelfieAppStore），然后点击Generate。

下载profile，然后去文件夹中打开你刚刚下载的provisioning profile。会以*.mobileprovision*结尾的两个profile（见图10-19）（MySelfieAppDev.mobileprovision和MySelfieAppStore.mobileprovision）。双击这两个profile即可添加到Xcode，添加成功后，就可以在工程中看到这两个文件了。



Page 270 | Chapter 10: Running on a Device

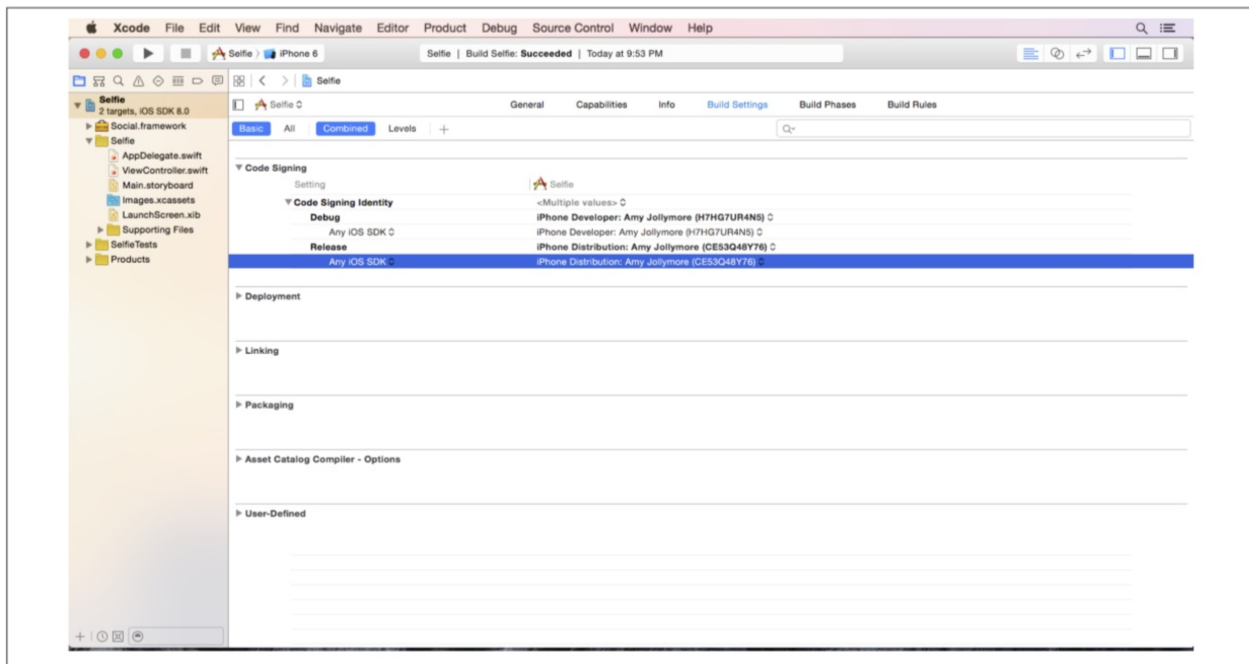
打开Xcode工程，点击Project Navigator中的工程名字，Editor中会显示出工程的详细信息。在Team里的下拉菜单中选择Add an Account。登录你开发者帐号和密码，接着点击Add（见图10-20）。



Profiles | Page 271

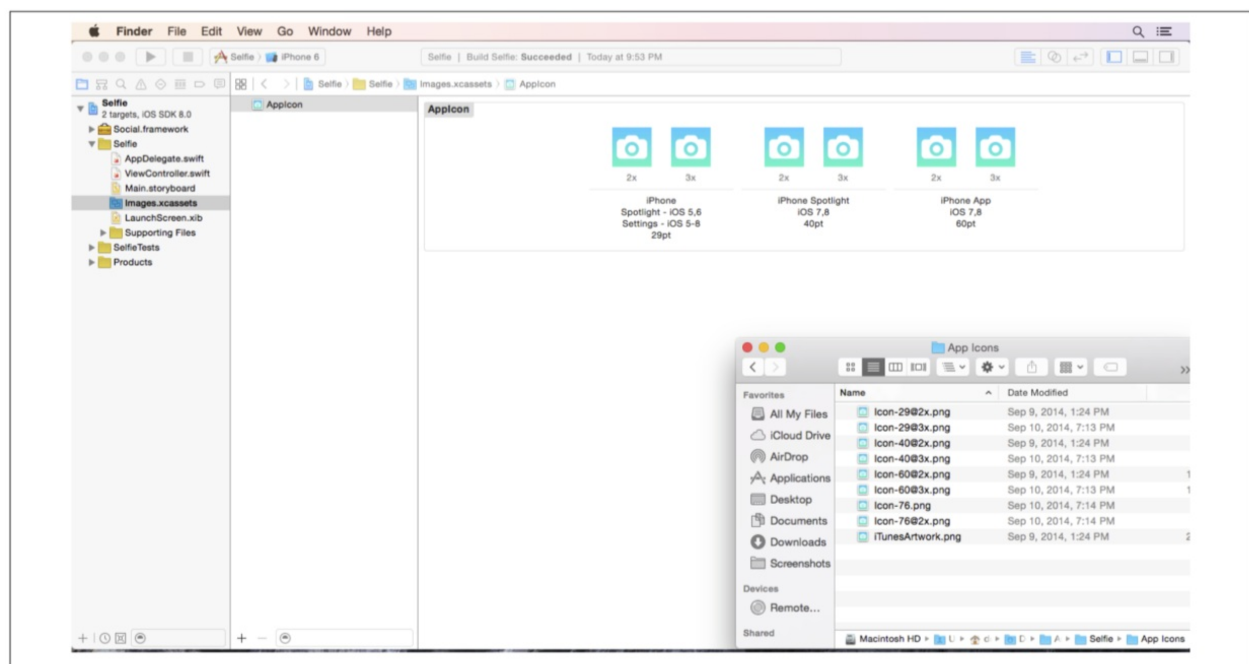
关闭Accounts对话框，回到工程详细信息中来，点击上方名为Info的tab按钮，清空Bundle Identifier，然后输入上创建App ID时输入的倒序Bundle ID（例如：*com.johnsmith.selfieapp*）。

接着选择Build Settings，滑到Code Signing区域，在Code Signing Identity下方，Debug一栏选择iPhone Developer：你的名字。确保下方所有的iOS SDK都是同样的名字。Release一栏选择iPhone Distribution：你的名字。确保所有的iOS SDK都是同样的名字（见图10-21）。



点击上方General这个tab选项，确认bundle和团队信息是否正确无误。

打开浏览器，到AppSchool下载App icons（图10-22）。（这个网址我一直打不开，我也不知道咋回事，一开始以为是被墙了，后来觉得可能是网站已经关闭了吧）



> Profiles | Page 273

打开存放icons的文件夹，然后打开Xcode，把文件夹中的图标拖动到Xcode中的*Images.xcassets*里。

现在，你可以在真机上运行你的程序了，把你的iOS设备连接到Mac上，点击Xcode顶部iOS Simulator下拉菜单，选择iOS device，点击Play按钮（Run）。

如果你在真机上运行我们之前讲解的小应用Selfie，你可能会看到Xcode的警告“process launch failed: Security”，这是第一次运行时的警告，打开你的设备（iPhone或iPod等等），点击Trust from the App Developer，接着App会向你获取使用摄像头的权限。点击Home键，回到桌面，你看会看App的图标已经显示在桌面上了。

祝贺你，现在，你已经成功地在真机上运行应用了。

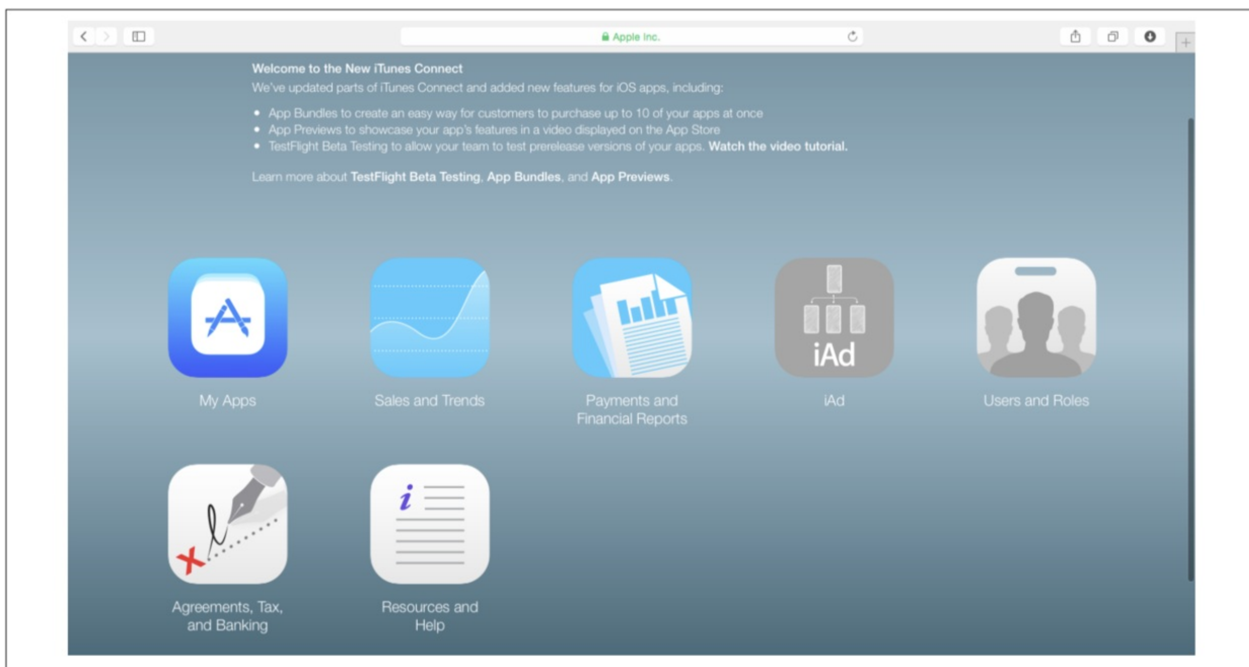
如果你收到了错误或者警告，不要担心，熟能生巧，到网站上下载源码（这个网站可能废掉了），对比学习，多练习几次，直到成功~

> Profiles | Page 273

第十一章：提交到应用市场

在本章节，你将学会如何将App提交到App Store，还将学习使用iTunes和苹果应用商店的管理软件（见图11-1）。本章节将引导你到iTunes Connect，因此你手边必须要有一个Mac。

> 
> 这个App提交到应用商店只是出于学习目的。 它是不可能被批准可以进入应用商店的。 提交App的时候，你要确保使用一个独一无二的应用名字和Bundle ID。

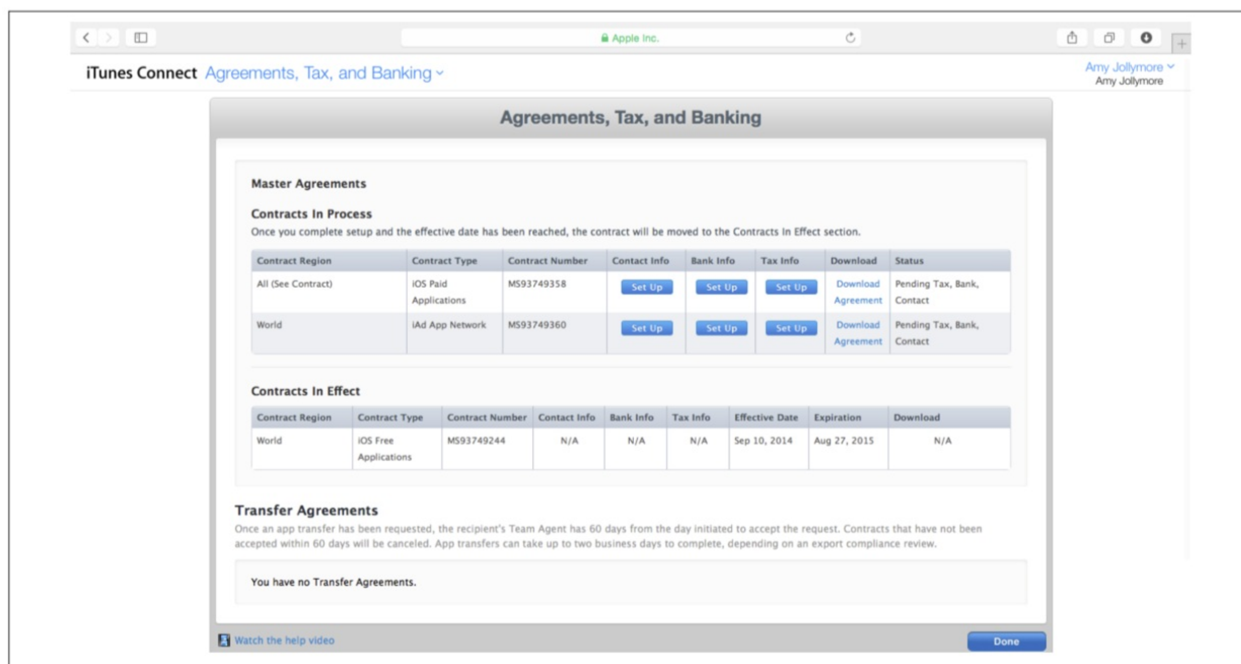


> Page 275

iTunes Connect是用来管理应用市场中的应用。这个网页有所有的营销、报告、合同信息。在http://itunesconnect.apple.com使用开发者帐号登录iTunes Connect。iTunes Connect在应用商店会提供一个叫iTunes Connect Mobile的配套应用。

协议，税务和银行Agreements, Tax, and Banking

将一个app提交到应用商店以前，你需要先完成“Agreements, Tax, and Banking”这部分（见图11-2）。它管理着苹果公司和开发者之间的合同。它记录着开发者银行账户信息。



默认设置下，iOS开发者计划允许所有开发者在应用商店免费向全世界发布免费的应用程序。但是，如果你想要在应用商店出售你的应用程序，你就需要一个单独的合同。iOS Paid Applications合同可以让开发者在应用商店出售应用。

要同意附加的合同，需要在iTunes Connect的主页点击“Agreements,Tax,and Banking”选项。在Request Contracts部分，在Paid Applications line中点击Request按钮。阅读合同条款，决定你是否同意合同中的条款。如果你同意条款，点击Submit，点击Done。

> Page 276 | Chapter 11: Submitting to the App Store

点击Setup Contact Info按钮来生成公司合同，输入你的全名，地址，邮箱，以及电话号码。把这个公司合同设置为“Senior Management, Financial, Technical, Legal, and Marketing”，点击Done。

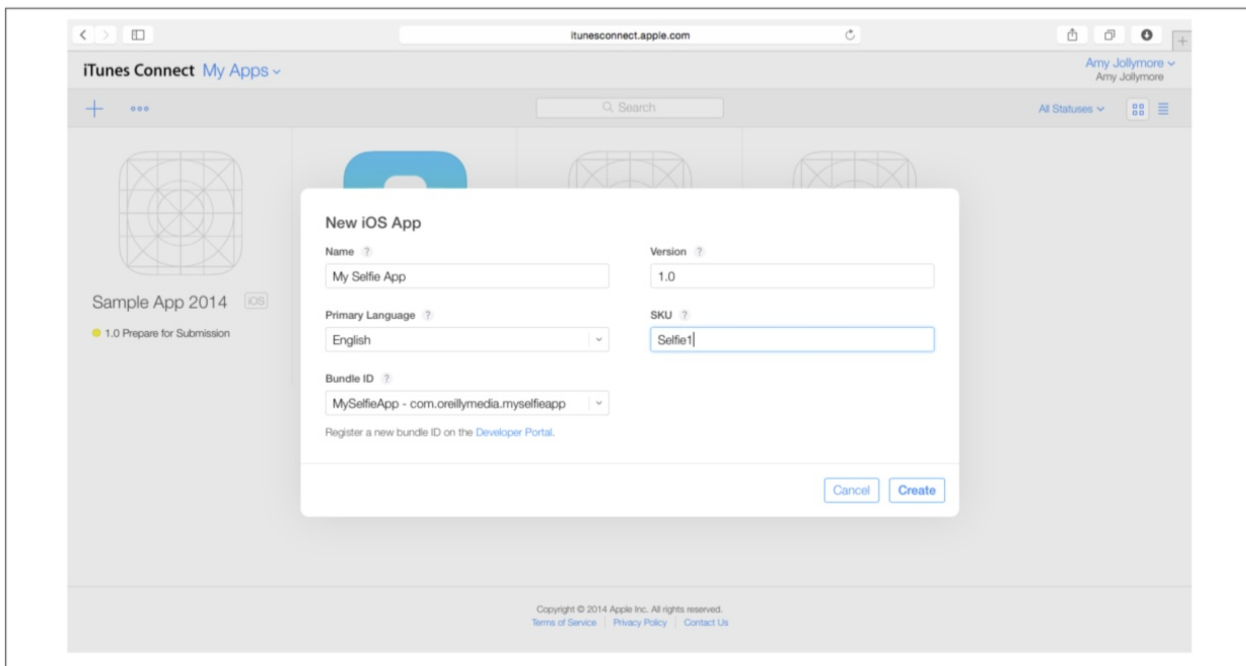
点击Setup Bank Info按钮，添加你的银行信息，点击Add Bank Account，输入国家，输入银行的的ABA routing number（汇款路线号码）。如果你不知道这些信息，点击下方的lookup按钮。最后，选择银行，输入银行帐号。你的收入每个月会直接打入到你的银行帐号里。

最后，点击Setup Banking and Tax Info按钮，点击U.S. tax forms（美国保税表格），输入的合法姓名，如果是公司，输入法人实体。选择获益人的类型，如果你具有免税资格，你需要咨询一下你的会计接下来该怎么做。提供你的合法地址和合法名字。在纳税人号码的那一栏，提供你的社会保障号码或者EIN。仔细阅读认证部分。这部分一旦提交是不能够被修改的。在继续下面的部分之前一定要确定提供的信息是正确的。

现在“Agreements,Tax,and Banking”部分已经完成，是时候给App Store创建一个应用程序列表。这个应用程序列表包含了这个App所有的市场营销信息，包括名字，描述信息，图标，关键字，截图等等。

创建应用列表Create App Listing

My Apps这部分管理应用列表，从iTunes Connect主页点击My Apps，点击左下角的加号按钮，第一个屏幕展示需要填写的应用程序的基本信息（见图11-3）：Name 填写你想要展示在应用市场上的名字。 Version 在这里输入应用版本号，通常1.0表示这是一个新的应用。Primary Language 这个是应用列表的显示语言。 Bundle ID 从下拉菜单选择一个可用的App ID，一旦创建完成应用，这个ID就无法修改。 SKU 这部分是你喜欢的，用来识别你的App。



Create App Listing | Page 277

每个版本的应用在应用商店的都有自己的数据。一旦应用程序发布了，这些信息的大部分是不能被修改的。在一个新的版本发布后，就可以更新应用程序列表中的信息了。输入你的应用名字；确保它不是Selfie。App应用商店不允许应用出现两个完全相同的名字。看看下面这个例子：

Selfie - Take and Share Selfie Photos

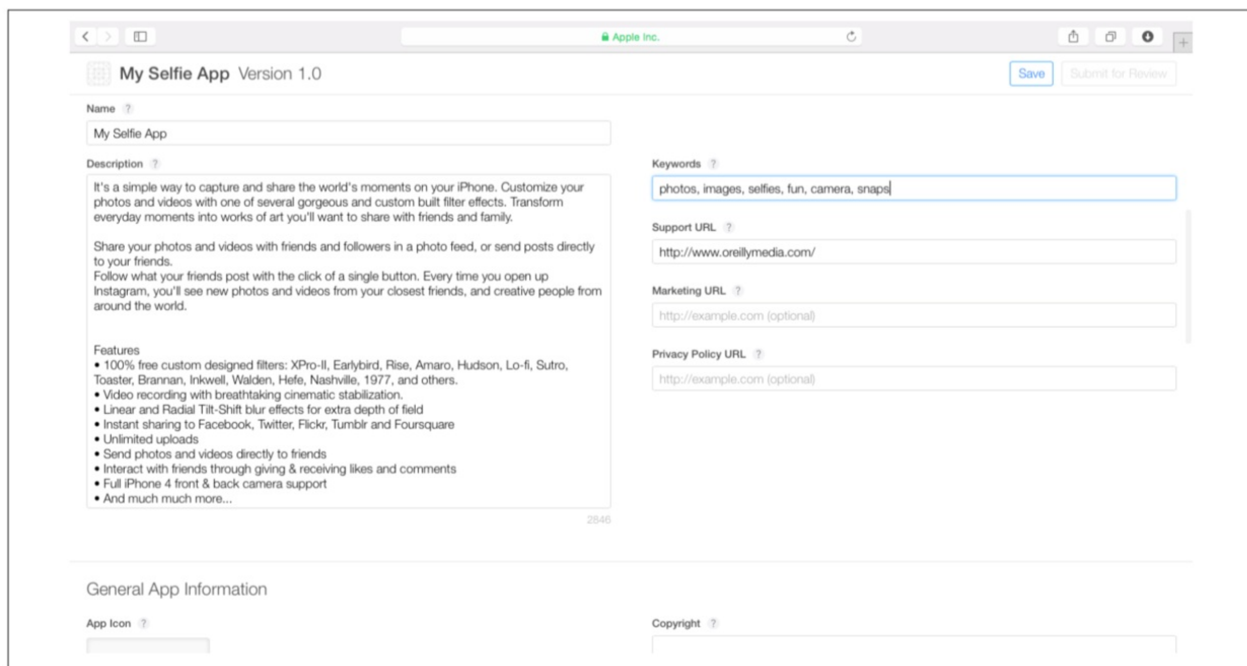
对于SKU，输入一个独一无二的代码去识别你的应用程序。你可以将你的姓和名字与SKU数字结合在一起。比如：

JOHNSMITH001

然后选择在前面的章节中创建的selfie ID包，可以从下拉菜单中找到。点击Continue。

版本信息Version Information

接下来，出现应用列表页。向下滚动，注意到一个叫版本信息的部分。这个版本信息具体到应用程序的每一个版本。只有在应用程序更新时，它里面的大部分信息才可以修改。



Page 278 | Chapter 11: Submitting to the App Store

Version Information部分收集下列信息： 版本信息部分收集了以下这些信息

App name 出现在应用商店中的应用名称。 Description 在这里你需要提供细节以及说明用户为什么要下载你的应用程序。 Keyword 这些关键字将帮助用户通应用商店搜索找到你的app。 Support URL 这个网址可以让用户反映问题和获得反馈。 Marketing URL 是一个可选的网址，提供更多关于应用程序的信息。 Privacy Policy URL 是一个可选的网址，详述应用的隐私政策。自动更新的订阅应用和目标群体是小孩的应用必须要有这一项。

关键词Keywords

当用户在应用商店中进行搜索时，显示的关键词列表会帮助用户找到你的应用。关键字用逗号分隔开，之间没有空格。输入的关键字的数量限制在100个字符内，逗号也包括在内。这些关键字不能使用其它应用程序的名字或者品牌。好的关键词能够简单描述应用，可以解决什么问题。比如：

Create App Listing | Page 279

photos,image,snapshot,pictures,photograph,text,label,portrait

创建你自己的关键词列表，放入keyword输入框。

开发者网站Support URL

用户可以在这里提交问题、排除解决问题，网页上有联系信息或者客服邮箱会更有帮助，你可以在http://www.tumblr.com创建一个免费的网站。

在Support URL输入框中输入Support URL。

描述Description

描述应该提供应用的简要概述和工作原理。在这里写下说明；借鉴相似应用的描述。这里是一个很流行的照片共享应用的描述，Instagram:

超过两亿的用户爱上Instagram！在你的手机里，它可以让你的拍摄和分享变的非常简单。定制设计你的照片，使你的video拥有一个华丽和个性化的滤镜效果。当你遇家人和朋友在一起的时候，你可以使用Instagram将每天的时刻变得更加艺术化，并与他们分享这些。

```
Instagram 是一种捕捉和分享世界每一精彩瞬间的简单方式。
你可以每天拍些照片和视频，将它们转变为精美的艺术与亲朋好友分享。

只需关注，即可借他人之眼来捕捉世界的瞬息变化，关注对象不仅限于认识的人，
还可以是有灵感的 Instagram 控、摄影师、运动员、名人等流行符号。
每次当你打开 Instagram，就会被挚友的新照片和视频，
以及世界各地冒险一族的惊人瞬间给牢牢吸引。

超过 4 亿用 Instagram：

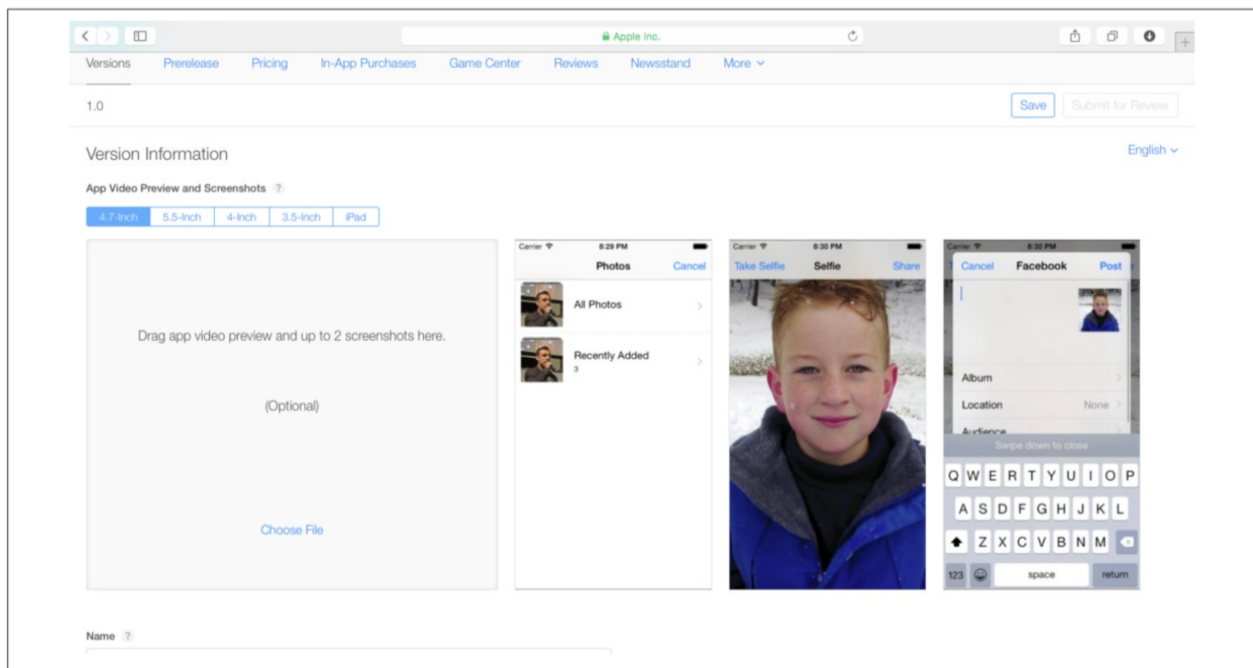
* 使用可定制滤镜（免费）编辑照片和视频。
* 使用 10 种高级创意工具美化照片，调整亮度、对比度、饱和度、光影、高亮和视角。
* 根据帐户和赞过的照片，关注他人。
* 使用 Instagram 可定制稳定化功能，让视频具有影院画质。
* 立即分享照片至 Facebook、Twitter、Tumblr 等社交网络。
* 与全世界的 Instagram 用户建立联系，关注对方的照片和视频更新。
* 通过私信，向好友直接发送照片和视频。
```

输入你的应用概述，确保概述涵盖了应用的核心特性。为什么这个应用有用？能用它来做什么？是为哪些人设计的？

> Page 280 | Chapter 11: Submitting to the App Store

截屏Screenshots

截图部分按理说应该是营销应用最重要的部分（图11-5）。研究表明大部分iOS用户会直接看截屏，从而决定他们是否去下载这个App。你的截屏应该鲜亮，充满色感。至少每一张截屏应该具备4.7英寸，5.5英寸，4英寸和3.5英寸这几个尺寸。如果应用程序是通用的，那么iPad的截屏应该需要的。如果你的应用只有iPad版本，就无需提供iPhone尺寸的截屏。



出OS模拟器生成截屏的方法：启动应用，按下Command＋S，一个截图就会出现在桌面上了。

为iPhone4s，iPhone5s，iPhone6，iPhone6Plus截图。然后回到iTunes Connect。

确保4.7英寸被选中，然后从桌面将iPhone6的截图拖动到iTunes Connect页面中。所有的截图上传完毕后，你可以用拖动的方式来排序。第一张截图会出现在应用商店搜索结果中。

选择5.5英寸的屏幕然后将iPhone6 Plus的截图拖进去。选择4英寸的屏幕然后将iPhone5s的截图拖进去。选择3.5英寸的屏幕然后将iPhone4s的截图拖进去。

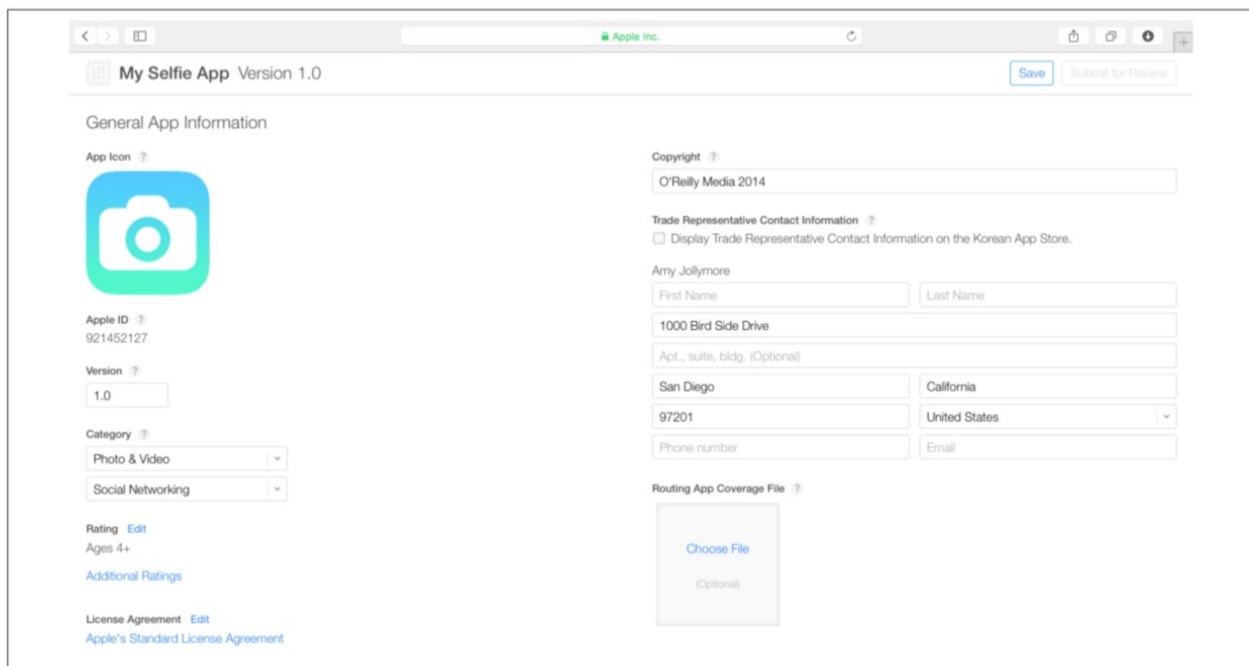
完成这些操作后，点击iTunes Connect右上角的Save。

> Create App Listing | Page 281

应用的大概信息General App Information

接下来这部分，叫General App Information，需要下列信息：

App Icon 应用图标用于应用商店的展示。提供1024*1024的图片，格式可以是JPG，TIFF，PNG。图标中不允许用透明，也不要圆角。 Category 为你的app选择第一和第二分类。 Trade Rep Info 此联系信息用在韩国的应用市场里。 Routing App Coverage File 可选的，用于指定应用支持的地理区域。 Version Number 标识每个版本，格式为1.0，1.1，1.2等等。 Copyright 每个人或者公司拥有属于自己应用程序的版权。 License Agreement 一般默认使用标准的用户许可协议。如果你想改变这个，请点击位于app listing底部的License Agreement旁边的Edit按钮。



> Page 282 | Chapter 11: Submitting to the App Store

小贴士Tips

当你打算营销策略，需要记住下面几条：

1 像开发应用那样努力营销你的应用。 2 要有引入注目的应用图标和截图。 3 看一下销售榜单上的应用，借鉴它们，然后设计你的图标和截图。 4 应用描述应该简洁扼要。向你的消费者解释你的App能够解决什么问题以及为什么它是最佳选择。描述应用核心特点，你的App是如何使它的使用者受益。 5 还有一个小贴士就是，你可以尝试为你的应用使用一个新的名字，分类以及关键词。这个简单的修改可以增加你的下载量。 6 时刻关注应用商店的最受关注的部分。 7 阅读你App的评论，并且根据App用户的需求进行迭代。

> Create App Listing | Page 283

应用图标App Icon

应用程序图标是你营销工具中最强大的一个工具。当用户在你的app和别人的app中选择时候，好的应用程序图标是一个决定性因素。在AppSchool.com/book中下载你可以用到的图标。然后点击Choose File按钮。

选择这个iTunesArtwork.png的图标文件。这个图标是1024*1024的像素，它将用于App Store。

分类Category

选择一个正确的应用商店分类也是营销计划中重要的一个环节。把你的App划分在一个合适的分类中，这将会吸引更多的用户去关注你的App。

因为Selfie是一个基于照相机类的App，它首先被划分在Photo&Video类。在下拉菜单中看一下其它的选项，去选择第二个分类。

评级Rating

评级信息用于指导家长和用户获知此App中的内容安全等级。苹果公司要求你按类型填写App内容的等级。例如，苹果公司要求标注你应用内暴力展示的暴力等级，Selfie这个App不具有任何暴力，设置为None。

点击Rating旁边的Edit按钮，评估一下，然后选择 None，Infrequent，或Frequent。

贸易代表联系信息Trade Representative Contact Information

Trade Representative Contact Information仅用在韩国的苹果应用商店，根据韩国的法律，可供下载的应用必须要符合韩国的法律。下列的信息只用于韩国的苹果应用商店：

输入下列信息：

- First name（名字）
- Last name（姓）
- Email（邮件）
- Address（地址）
- City（城市）
- State（声明）
- Postal code（邮政编码）
- Country（国家）
- Phone number（电话号码）

> Page 284 | Chapter 11: Submitting to the App Store

版权Copyright

版权是指的个人或企业法人在当前拥有此应用程序的权利。

在copyright输入框中输入你的名字。

应用审查信息App Review Information

这个信息只发给苹果的审核团队（见图11-7）。审核团队可能会就App的某个特性练习你，所以要输入你的联系方式。

> Create App Listing | Page 285

输入下列信息： First name 名 Last name 姓 Phone number 手机号 Email address 电子邮箱 Demo account 模拟帐号/示例帐号

示例帐号/模拟帐号**Demo Account**

在某些情况下，审核你的应用需要用到一个示例帐号，如果App需要登录，最好提供一个帐号以供审核团队成员测试。这样能让他们看到你应用中所有的特性，以提高通过审核的可能性。

在Selfie这个应用中，我们不填写的Demo Account。

备注**Notes**

备注部分用来告诉审查人员某些事情，以防他们可能没有注意到，在使用应用时他们应该知道的某些事情。除了应用描述外，这里可以提供App的额外细节。

我们开发的Selfie，可以空着这一栏不写。

版本发布**Version Release**

接下来部分是Version Release，询问应用通过审核后是否立即上架应用商店，如果不是，那么你可以手动上架应用，或者设定某个时间上架。

在Version Release勾选框勾选上Automatically Release this Version。

语言Languages

应用商店当前有28种语言，如果可能，最好适配每种语言。如果没有适配某种语言，此语言的用户就无法看到当地语音的应用列表信息，他们只能看到英语的应用列表信息，他们不一定懂英文。

滑到上方点击English，从下拉菜单中查看其他可用的语言。

> Page 286 | Chapter 11: Submitting to the App Store

定价Pricing

重新滑到上方打开Pricing这个tab页，接下来需要你填写定价信息（见图11-8）。

Availability Date（上架时间） 如果你打算让你的App在特定的某一天上线，请在此设置日期。

Price Tier（产品价格） 设置APP的售价。苹果公司会从销售额的收入中收取30%的提成。

Price Tier Effective Date（价格有效时间） 使用有效时间来设置一个临时的价格。

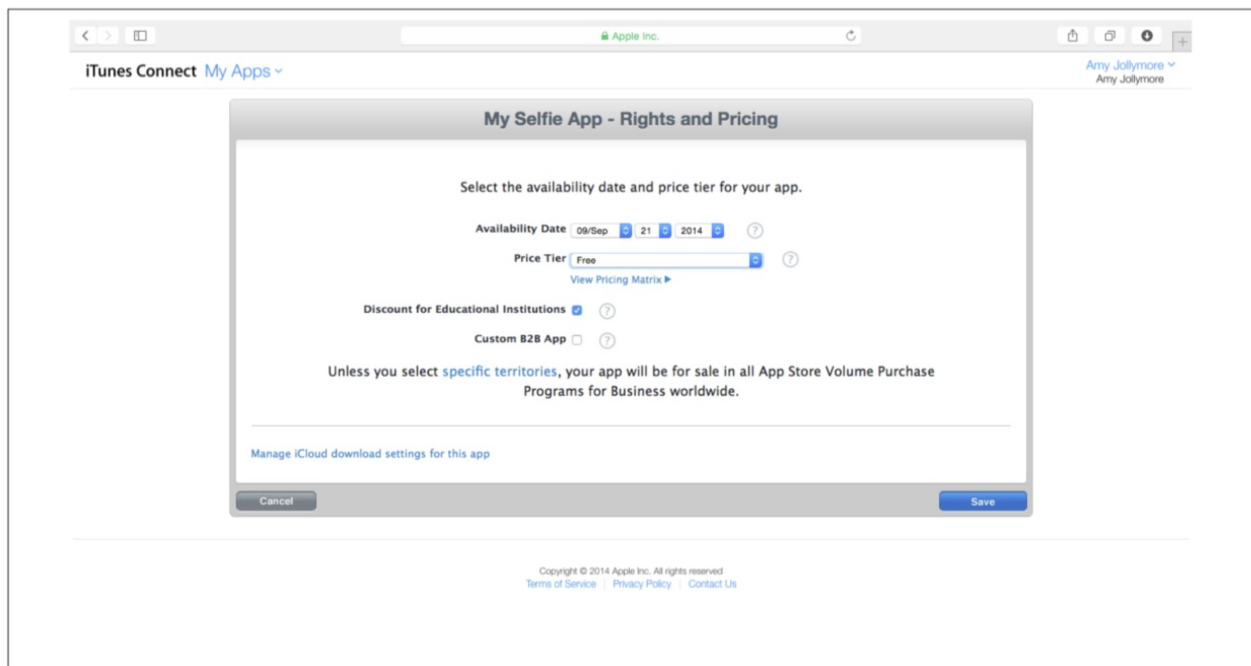
Price Tier End Date（价格结束时间） 可以设置新价格的时间。

Discount for Education Institutions(为教育机构提供打折) 教育机构批量购买将会得到折扣。

Custom B2B App（定制B2B应用程序） 可选项，你的应用是否是B2B的应用程序。如果是B2B的应用，那么该将无法公开出现在应用商店里。

Territories（区域） 哪些国家和地区能够在应用程序商店获得此App。

iCloud Download Setting（iCloud的下载设置） 此选项决定该应用的哪个版本可以从iCloud中下载。



> Create App Listing | Page 287

如果你打算让你的app在特定的某一天上线，请把availability date设置为将来的日期。举个例子，你发布App的时间应该和营销活动呼应。

如果你想让App通过审核后立即上线，那么把availability date日期设置为今天。

Price Tier管理下载应用的售价，以美元为基础，区间1是$0.99，区间2是$1.99，区间3是$2.99，在其他国家会收取对等价格的当地货币。

Tier0表示此应用免费。

教育机构批量购买可以得到折扣是别以为他们是批量购买的，如果你允许他们获得折扣，请勾选此选项。定制的B2B企业应用只能用于Volume Purchase Program用户，不能出现在应用商店中。

因为Selfie这个App是免费的，也就没有什么折扣可提供了，所以不勾选此选项。

App能够在世界范围内超过150个国家中下载，如果你想要限定App仅限于某个国家，请选择页面底部的Specific Territories链接。

我这里，我们的这个Selfie应用不选择特定的国家，直接点击Save，接着点击Cancel。

上传二进制文件Uploading Your Binary

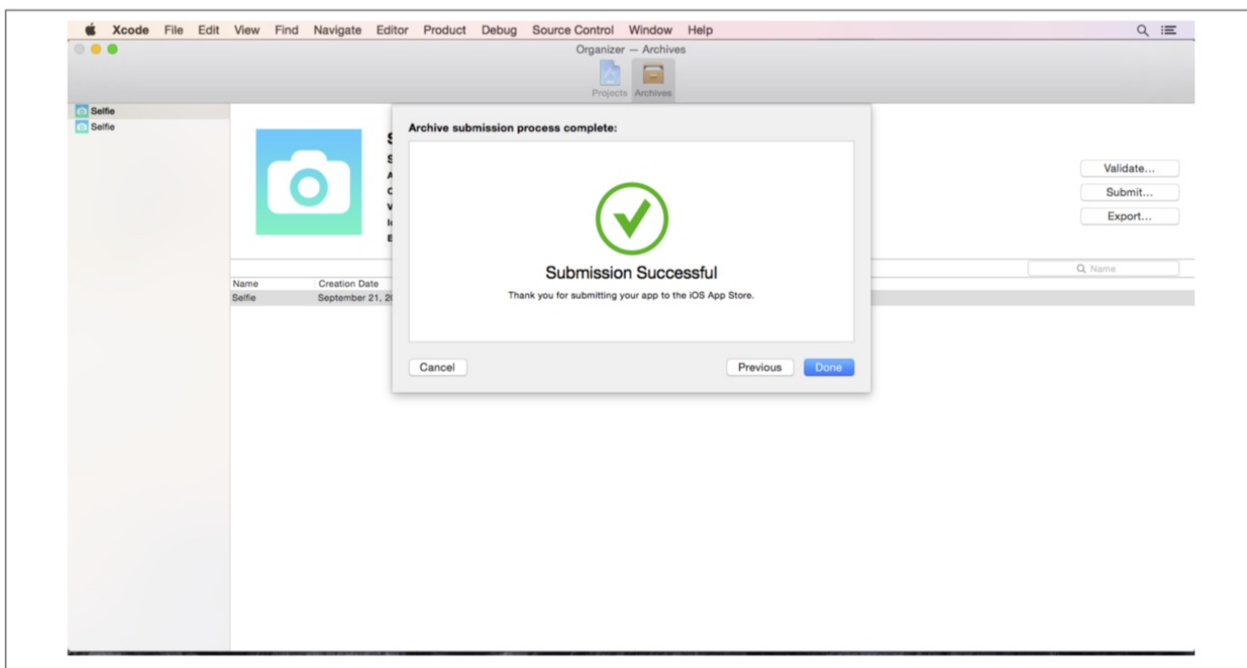
填写玩App清单后，点击Save。接着，上传二进制文件，也就是你的App。

> Page 288 | Chapter 11: Submitting to the App Store

打开Xcode的工程文件。在顶部的工具栏，点击设备模拟器的下拉菜单，选中iOS Device。这样做是为了确保应用程序是在一个真实的设备中运行，而不是在模拟器中。接下来，从顶部菜单栏，点击 Product—> Archive。

这样就能使用证书和provisioning profiles创建好应用程序了。创建完毕后，出现Organizer，点击顶部菜单栏Window->Organizer设置发Organizer。点击Archives这个tab按钮，出现archives列表。

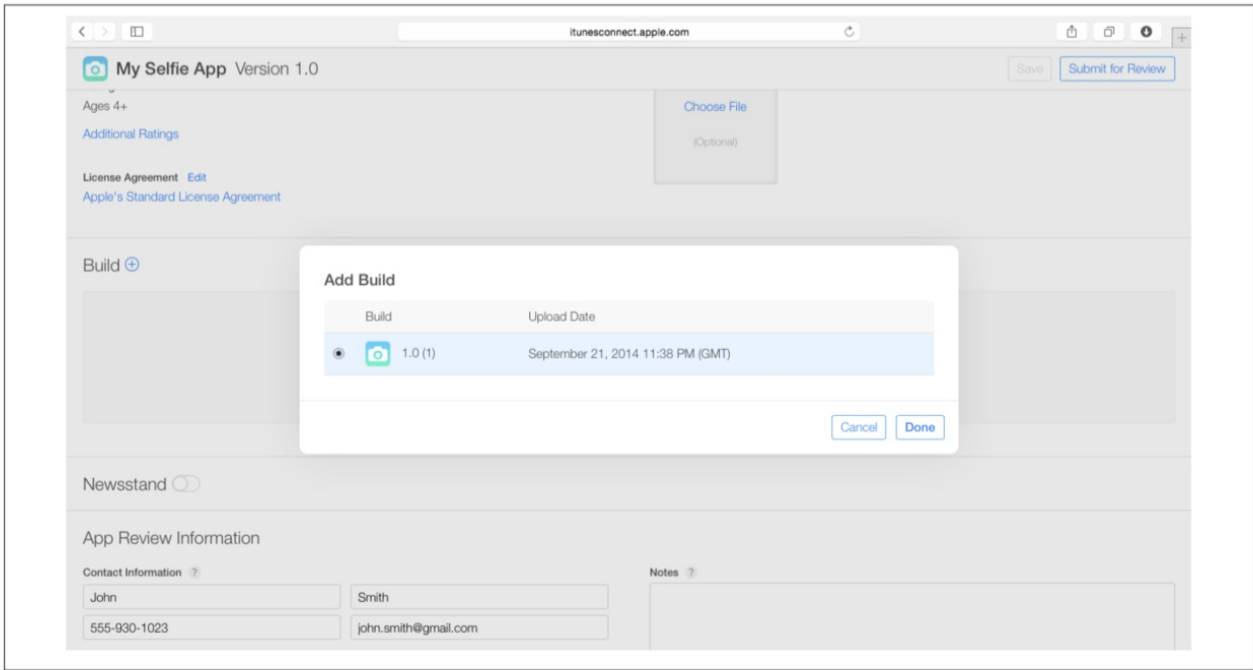
点击存档的应用，然后点击窗口顶部的Submit按钮。接着提示需要提供你的Apple ID和密码。选择应用程序的名字和应用商店的provisioning profile。点击Submit。这个应用程序将被分析上传到iTunes Connect（见图11-9）。



The Build Section

在Build Section这部分你需要把你上传的应用和刚刚填写的应用清单App listing匹配一下。

打开iTunes Connect中的应用清单列表，滑倒底部找到Build Section，点击加号按钮，选择uploaded binary（见图11-10），如果你没有找到这个加号按钮（Click + to add a build before you submit your app），等三到五分钟，刷新此页面。Build Section的信息更新可能会有延迟。点击Done，滑倒顶部，点击Save。

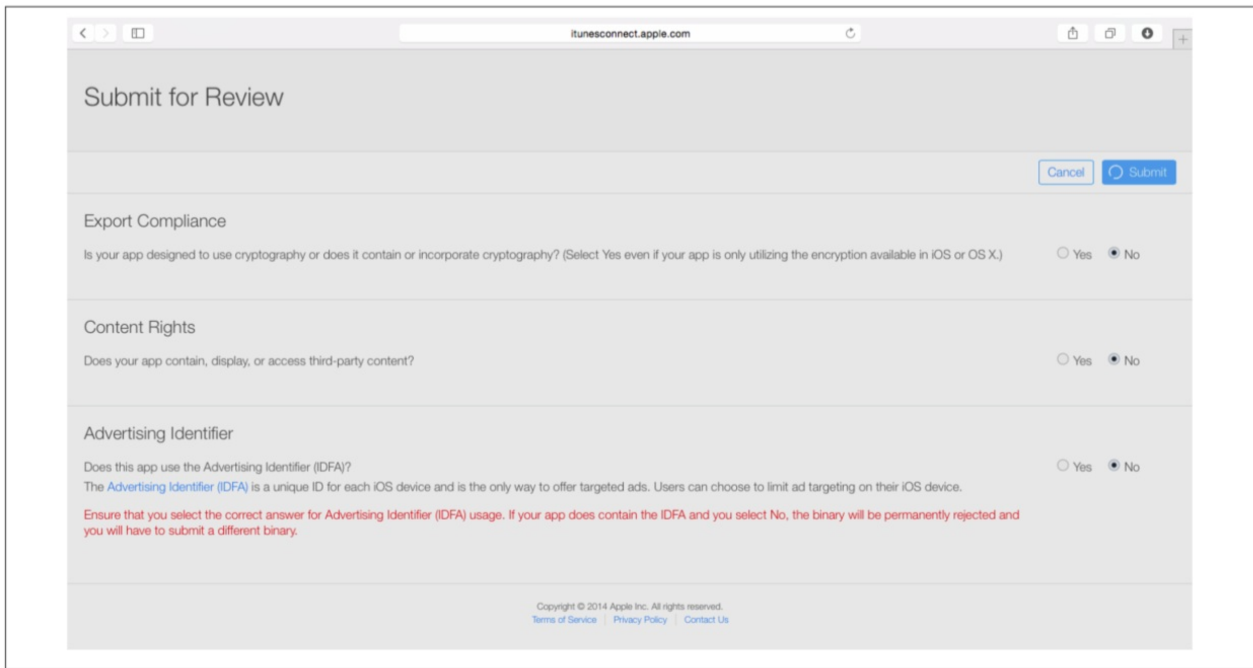


Create App Listing | Page 289

准备提交Ready for Submit

现在，应用程序的清单完成了。滚动光标到顶部，点击Save，然后点击Submit for Review。

出现一堆问题（见图11-11）。根据你的应用程序的情况回答每一个问题。Selfie没有使用cryptography或者incorporate cryptography，Selfie不包括第三方的内容，Selfie没有使用广告。回答每一个问题然后点击Save。



Page 290 | Chapter 11: Submitting to the App Store

状态Statuses

提交后，App的状态变为“Wating for Review”（等待审核）。这里有一个列表，包含所有的状态。一般不会经历所有的状态就可以上架，在大部分情况下，App会从“Prepare for Submission”（准备提交应用）到“Waiting for Review”（应用等待审核），再到“In Review”（正在审核中），最后到“Processing for App Store”（提交到应用商店）或“Rejected”（拒绝上架）。页面下方的Status History可以查看应用的所有历史状态：

Prepare for Submission（准备提交应用） App已经在iTunes Connect中创建完成了，但是还没有提交给审核团队审核。这时，应用清单列表中的信息还是可以修改的。

Waiting for Review（应用等待审核） App已经提交给审核团队审核了，审核需要用多久时间无法确定。这时，仍然可以修改应用清单列表中的信息。

In Review（正在审核中） App正在审核，审核需要用多久时间无法确定。这时，就不可以修改应用清单列表中的信息了。

Pending Contract（合同待定） App已经通过审核了，但是还没有生成合同。应用清单列表中的信息不可修改。

> Create App Listing | Page 291

Pending Developer Release（等待开发者发布应用） App已经通过审核了，但是需要等待开发者手动发布。应用清单列表中的信息不可修改。

Processing for App Store（提交到应用商店） App已经上传到应用商店了，24小时内就可以从商店中下载到你的应用了。应用清单列表中的信息不可修改。

Pending Apple Release（等待苹果发布应用）（这个地方我觉得可能翻译的不对，欢迎留言告诉我正确的译文） 苹果公司保留你的应用直到发布新的iOS系统。确保应用设置的目标平台有效。应用清单列表中的信息不可修改。

Ready for Sale（可以进行销售） 应用可以在商店中下载到了。应用清单列表中的信息不可修改。

Rejected（拒绝上架） 苹果公司拒绝了你的应用。打开Resolution Center获取更多的详细信息。应用清单列表中的信息不可修改。

Metadata Rejected（元数据被拒绝） 这说明你的应用清单列表没有通过审核。修改清单列表中的信息，重新提交审核。应用清单列表中的信息不可修改。

Removed from Sale（苹果公司下架应用） 应用被苹果公司从商店下架，苹果公司在下架你的应用之前会联系你。应用清单列表中的信息不可修改。

Developer Rejected（开发者拒绝上架） 开发者，也就是你，拒绝App商家。应用清单列表中的信息不可修改。

Developer Removed from Sale（开发者下架应用） 开发者，也就是你，把应用从商店中下架了。应用清单列表中的信息不可修改。

Invalid Binary（无效的二进制文件） 你上传的文件不符合要求。应用清单列表中的信息不可修改。

应用审核App Review

App的状态变成In Review后，苹果公司的审核团队就会审核你的应用。审核过程可能是几个小时，或者几天。审核是为了确保你的应用没有bug，应用内容与你填写的应用描述相符合，同时符合Human Interface Guidelines和App Store Guidelines中的要求。

> Page 292 | Chapter 11: Submitting to the App Store

应用驳回App Rejection

应用被拒也是学习开发的一部分，不要把被拒当成一件坏事，当做一次学习的机会。可以在Resolution Center中直接和苹果的审核人员沟通，可以问问题，获得更多如何修复问题的详细信息。唯一不好的是，被拒后重新提交审核，你又排到了等待审核队伍的最后面，所以又要再次等待审核。

通过审核App Approval

当App的状态变成In Review时，你会收到一封邮件。大部分的审核会在审核当天完成，但是总有例外。大部分的开发者会估计一周的时间用于审核。审核通过后，App的状态变成Processing for App Store。一般24小时左右就会出现在应用商店中。App的状态变成Ready for Sale后，你会收到iTunes Connect的邮件。

> Create App Listing | Page 293

第十二章：管理和推广你的应用

在这一章，你将学会如何追踪下载量，查看销售数据，更新你的App，改善你的营销策略。这些技术能够将你的应用提高到更高的水平。

追踪销售和下载量Tracking Sales and Downloads

把你的App发布到应用市场并不代表着你的工作到此结束了，一旦App批准上架，那么需要我们合理地管理和营销App，监视下载量和销售量是App中最重要的衡量指标，你可以使用iTunes Connect里的Sales and Trends部分来查看。

Sales and Trends部分会在PST上午八点进行更新每日的数据，当年打开Sales and Trends部分时，会出现一个大图表，标注了总的下载量，以时间段分割开来。点击Proceeds这个tab按钮就可以查看销售收入，图标也会变成销售收入图表，你还能够查看你的App最受欢迎的地区国家，这些信息能够帮你做出一些更新翻译决策，在Sales and Trends中Platform部分会展示出来哪些类型的设备曾经下载过你的App，你也可以把这些数据导出来，是csv和xls格式，点击小的下载按钮即可下载。你可以按日、周、月、年、以及整个生命周期的数据报告，点击左上角的下拉按钮，选择Reports即可下载。

支付信息和财务报告Payments and Financial Reports

iTunes Connect每月会提供一个总收入和支付的报告，这个报告比起Sales and Trends来说更精确一些，但是只能每个月计算一次。你可以查看两年内的支付历史记录。苹果公司也提过详细的收入报告，这些报告可以在年底保税的时候提供一些帮助。

> Page 295

点击iTunes Connect主页的“Payments and Financial Reports”可以查看收入和支付报告。

点击Summary这个tab按钮，会显示之前月份的收入报告详细信息，包括每个国家的App销售收入，当然“Payments and Amounts Owed”部分也会展示最后的收入和总收入，你可以使用屏幕下方的Trends图表按月标注和追踪你的支付信息。

点击Earnings这个tab按钮，会出现每个国家收入详细信息，可以按月下载。国家的收入会以当地的货币展示出来，点击Download下载财务分析报告。点击Owed这个tab按钮会展示每个国家的总收入，然后转化成你的当地货币。这个总收入会列在屏幕的右上角，这个数字并不是最后的最终结果，在支付之前可能会有一些变动。所有的App都要给苹果30%的佣金。

Payments这个tab页显示的详细的收入、税收和汇率。支付发出三十天后完成，例如，一月的收入会在三月的第一个周支付，二月的收入会在四月的第一个周支付。

崩溃Crashes

你的程序可能会有一些bug，没有按照你期望的情况运行，iTunes Connect给开发者提供了用户崩溃报告。这些崩溃报告可以用来定位确认应用中的bug。想要查看这些崩溃报告，我们需要打开iTunes Connect中的My Apps部分，接着点击你想要查看的应用，滑倒App列表的底部，在Additional Information下面，就能找到Cresh Reports。点击Download Report，下载扩展名为.crash的文件，双击打开文件，Xcode会分析这些崩溃信息。

评价Reviews

没有什么比听取你用户反馈更重要的了，在苹果的应用市场的评价系统里，用户可以提供一到五星的评价等级，同时可以输入文字评价内容。想要查看应用的评价信息，首先我们要打开iTunes Connect中的My Apps这部分，选择你想要查看的应用，然后点击顶部清单中的Reviews，然后所有的评价会按照版本和国家展示出来，我们可以用右上角的下拉菜单来改变想要查看的国家，阅读每位用户的评价，然后站在全局的角度来思考用户喜欢和不喜欢的地方。不要让某个消极或积极的评价就改变你App的初衷。然而用户评价仍然是你决定更新哪些内容的首要参考因素。

> Page 296 | Chapter12: Managing and Marketing Your App

更新应用Updating Your App

你的App上架后，你会获得更多的信息。为了让用户高兴，你需要不断改进App。之前下载过的或者购买过的用户都可以免费获得更新后的App。有很多原因让你更新应用，比如说，想要增加新特性，修复bug，想更新界面的交互体验，等等。给App提交新版本和提交应用到应用市场的步骤非常相似，所有App的更新仍然需要进行审核，通过后才能上架。

App Updates and MetaData Changes

想要给App增加新的版本，首先要打开ITunes Connect中的My Apps，点击想要更新的App，然后点击右上角的New Version按钮，屏幕出现新版本详细信息页面，填写新的版本号，如1.2，1.2等等，点击Create。然后填写App的信息页面，确保一定要填写“What's New in This Version”这部分，它会出现在应用介绍页面中Updates分页中。

大部分的应用介绍信息只能在App更新时才能修改。利用更新App的机会来修改App信息页面中的任何错误，如果有需要，还可以改变应用的关键词和名称。完成后，点击Save，“Submit for Review”，回答法律和加密问题，就可以上传新版本了。

优惠码Promo Codes

你的App上架后并不意味着人们就能够找到它，为了让潜在的客户找到你的App，苹果公司提供了优惠码功能，优惠码用来给付费应用提供免费账户，尤其是请求博客博主评价你的应用时，这优惠码就派上用场了。你可以给每个版本申请最多100个优惠码，申请优惠码首先要打开iTunes Connect中的My Apps这部分，点击你想要查看的App，滑倒底部，点击Promo Codes，出现优惠码界面，输入你要想生成的优惠码数量，点击Continue。查看协议点击统一，点击Continue，点击蓝色的Download按钮，下载下来一个扩展名为.txt的文件，这里有你需要的优惠码。

分析报告Analytics

iTunes Connect的Sales and Trends部分只能提供下载量和收入信息。要是能够了解用户是如何使用你的应用就好了，比如他们点击了哪些按钮？每次使用应用多长时间？用户在第一次打开App后还会再回来使用应用吗？所有的这些问题的答案都非常有价值。

> Updating Your App | Page 297

这些答案能够帮你了解到应用的具体表现，额外的分析由许多第三方机构提供。他们会在你的应用中放置一小段代码，然后分析公司可以开始收集用户行为信息了。所有的这些信息都是以匿名的方式手机的，这些分析数据比苹果公司提供的分析数据要详细多了。

下面这三个是提供免费分析服务的：

http://www.flurry.com http://www.google.com/analytics/mobile/ http://www.mixpanel.com

实名网址Vanity URLs

你可以给你的应用创建一个实名网址链接，实名网址用来给你的应用创建容易记忆的链接，在打印的纸质广告或者音频广告中特别有用。要创建链接，把公司名称或者应用名称添加到AppStore.com。例如，http://www.appstore.com/instagram会直接链接到应用市场里的Instagram应用。确保在创建链接时要去掉所有的空格用and替换。

还有一件事One More Thing

恭喜你！你做到了！我希望在这个从想法到应用市场的旅途中你收获了许多乐趣。但是旅程并没有结束。不断冲破你的知识边界，最好的学习方式就是研究一些你想要学习的东西。例如，你想学习更多关于Core Bluetooth？点击文档试一下示例代码。

记着，以后会有快乐的时候，也会有痛苦的日子，但到最后，你会觉得一切都是值得的。你有时候会止步不前，被一个bug困住，甚至觉得不可能修复了，这些日子是屈指可数的，所以要不断学习新的东西。永不放弃，在你知道它之前，你要笑对所有的事情。一旦解决了bug，你永远不会忘记自己是如何解决的。

Stay Hungry, Stay Foolish

> Page 296 | Chapter12: Managing and Marketing Your App

译本完结，说点收获

快一年了，终于完结了这本书。中间也放弃了好几个月。后来，阅读《自控力》，觉得需要实践其中的原理，就拿这个翻译当做我自己的第一个自控力实践项目了。翻译到一半的时候，明显的感觉到自己收获良多。

男友曾经说我免费翻译这个项目，没有什么利润，但还是在免费翻译，他算是明白了Github上的那些开源项目是咋来的了。我当时回了一句，可是我翻译这本书，得到的远远比我付出的要多得多啊。可得到了什么呢？还是来总结一下吧~

收获一：英文阅读水平突飞猛进

刚刚开始翻译的时候，我真怀疑当年大学六级是不是蒙着过线的。六级过了，当时没有让我具备阅读英文文献的能力，太无语。 从半个小时一页，到现在一个番茄钟翻译3页以上，提升了不少。现在我也不怕阅读英文原文博客、文献和书籍了。 多谢这次翻译项目，也谢谢《自控力》这本书。 英文水平算是我这次翻译项目最大的收获吧，毕竟通过这次翻译，我在学习编程时，使用的教程，都是RayWenderlich网站上的书籍和视频了，这之前可是想都不敢想用英文学习视频书籍教程，脱离中文字幕的生活会没有想象中那么可怕。

收获二：自控力增强了

从文章的发布时间可以看出来，我中间放弃了好几个月，自控力的问题一直困扰着我。尤其是高中寄宿之后，自控力就成了我最大的梦魇，甚至大学里，都无法摆脱掉。当时还陷入抑郁中，唉，不提也罢。一度不想念大学，打电话回家，说不想读了，读不下去了，现在回想，犹如梦中一般。 现在自控力虽然好一些了，并没有根治，只能一点点进步吧。比如最近用冰岛线织帽子，一开始不顺利，就放弃了。男友批评我，做事总是一开始兴趣大动力足，遇到了问题，或者失败了几次，就开始自暴自弃。让他这么一说，我又重新来过，最后，一个帽子用时一个半小时，在我看《芈月传》的时候，搞定了，一下子织了两顶帽子。

我使用的方法主要是《自控力》书中提到的几个小技巧。说实话，这本书我陆陆续续看了一年多了，但是总不见效果，在kindle中也是看了删，删了又复制回来，折腾了好几次。到了最后，我把书中的理论一概略过，把方法集合到一张小纸条上，当我要放弃的时候，比如翻译翻译着，突然想起有个美剧更新了，还没有看，我想去看美剧，可是翻译计划还没有完成。我就给自己规定，10分钟之后看美剧，然后手机定时10分钟后响起闹钟。这十分钟，我就看这小纸条，一个一个的看，看到哪里说不定我想试试，就做一下我看中的这个方法。等10分钟后，如果这个自控力方法起效果了，我应该已经又回到翻译上了。如果没有效果，我还是会去看美剧，很开心的去看，不是带着负罪感去看，接受自己去看美剧放弃翻译的行为，不责备自己。这样，能让我更快的回到下一个事情上去。

收获三：学到了一些编程知识

其实翻译这本初级入门书籍，是没有办法让我成为技术大牛的，毕竟这本书只是初级入门简单介绍的书籍。学完这本书，连个App都开发不出来，更不要提找工作了。还是需要学习更多的知识后，才具备开发App的能力。不过，当我回头看来，第一次看这本书，种种不明白，各种bug，万事开头难这话一点都不假。

收获四：获得了不少关注

也正是翻译文集，让我获得了不少关注，还有评论和喜欢。这些，都是对我的肯定。也许对方只是随手点击一下关注按钮，可是每次的关注提醒，都在提醒我，这翻译必须要有始有终了，不然如何对得起这么多关注的人，他们都是想看翻译文集，才会关注的。

收获五：算是我自己的一个里程碑吧

翻译过一本书，说出去也算是我今年的成就之一吧~过年的时候，也能过的心安一些。 给时光以生命，而不是给生命以时间。

收获六：认识道友

帮我翻译了第八章、第十一章，我来校对。没有翻译这本书之前，我的英文也就只能看个《小王子》，而今，我也担任校对工作了。

结尾的话

这本书毕竟是程序书籍，虽然初级，里面还是有专业知识的，如果在您在阅读中发现了翻译错误，请及时留言评论或者私信我，我会尽快更改的。

这次翻译，属于个人学习，严禁商用，请自行购买原版书籍。

就这么结束了，我的自控力实践项目一完结。 下个自控力项目，2016年见。（几天后见）